THE BEST SELLING COMPUTER PROJECTS MAGAZINE OCTOBER 1984

22 59 CHECK ON ORIC 72 28 SPLIT SCREEN

ELECTRONICS & COMPUTING

free inside every issue YOUR ROBOT

AN EMAP PUBLICATION

USA $2.95 Germany D6.00 95p

THE SUPERBASIC MYSTERIES

Nine clues to better QL programming

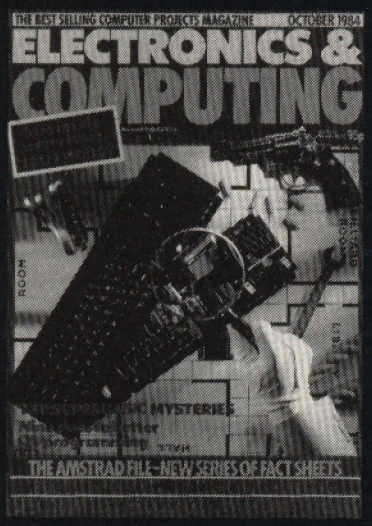
THE AMSTRAD FILE–NEW SERIES OF FACT SHEETS

BBC USER PORT UTILITY · BASIC BUG HUNTING

SPECTRUM TEXT PROCESSOR · CBM 64 SPLIT SCREENS

ELECTRONICS & COMPUTING Contents

Vol. 4 Issue 10



COVER PHOTO BY ROB BRIMSON

Electronics & Computing Monthly
Scriptor Court, 155 Farringdon Road,
London, EC1R 3AD

Editorial 01-833-0846
Editor Gary Evans
Deputy Editor William Owen
Production Editor Liz Gregory
Administration Serena Hadley

Advertising 01-833-0531
Advertisement Manager Anthony Herman
Chief Executives Richard Jansz
Francis Lee
Classified Tracey Keighley
Assistant Yvonne Moyser

Production 01-833-0531
Art Editor Jeremy Webb
Make-up Time Graphics
Publisher Alfred Rolington

Distribution
EMAP National Publications

Published by
EMAP Business and Computer Publications

Printed by
Riverside Press, England

Subscriptions
Electronics & Computing Monthly, (Subscription Department), Competition House, Farndon Road, Market Harborough, Leicestershire.

Electronics & Computing Monthly is normally published on the 13th day of each month.

PROJECTS

BBC Port Master 16
A handy utility that takes the hastles out of configuring the DDRs of the BBC computer's VIA.

EPROM simulator 28
A design that can dramatically reduce the time taken to develop machine code software eventually destined for EPROM.

Spectrum Wordprocessor 55
Complete listing of a versatile wordprocessor for the Spectrum.

Commodore 64 Split Screen 72
Cunning use of interrupts allows the screen of the '64 to be divided into two distinct areas, one displaying text, the other hi-res graphics.

FEATURES

Basic bug hunting 22
Mike James shows that a logical approach to hunting down program bugs can make the task far quicker and more enjoyable.

Bubble memories 34
Bubble memories have been hailed as the memory device of the future. To date, though, they have not made much of an impact on the microcomputer market. We assess the potential of this form of memory as well as giving a low down on just how they operate.

Interfacing the BBC micro 38
Paul Beverley continues his series on interfacing the BBC micro with a look at the operation of shift registers.

Extending QL BASIC 42
A number of BASIC programs that enhance the facilities of the QL are described together with some further ideas for increasing the power of the computer.

Amstrad CPC464 fact sheet 46
A collection of useful information concerning all aspects of the 464's hardware and firmware.

REVIEWS

FLEX in action 50
After last month's general introduction to the FLEX OS, we present further details of this powerful operating system that rivals OS9 for the affection of 6809 users.

Software file 79
This month our regular feature turns its attention to alternative languages for some popular micro computers.

PLUS

Editorial 10
News 10
Next Month 26
PCB service 36
Subscriptions 47
Book Service 77

And within *Your Robot*
Building low cost turtles, Pilot One's BBC computer interface and a further look at the Fischertechnick robot builders kit.

See page 60 for details of this month's competition.

Electron 64K? well, not quite

Sir Clive Sinclair caused quite a controversy when he launched the QL with the claim that it was a 32-bit computer but even this gamesmanship pales into insignificance when compared to one company that recently advertised in the Irish magazine Micro News and Market. The advert is headed with the bold claim that the Electron 64K is now available. 64K? Well, that's what the ad said! As no one in the *E&CM* office had heard that Acorn were producing an expanded version of the machine the advert naturally caused some considerable interest. A 'phone call to Ireland dashed all hopes of an *E&CM* scoop however as it turned out that the advert was for the 32K machine that we all know and love. The Irish distributer had, however, decided, in describing the memory size of computer, to include not only the RAM but also the firmware. Now while people are at liberty to do this the fact that conventionally only the RAM is included in the description of a computer's memory could prompt accusations of sharp practice. All the more so in view of the fact that those new to computing would be more than confused by such claims. A 64K Electron would appear to have more memory than a 48K Oric.

Let's hope that such practices do not find their way over to this country.

Amstrad in the lead

The Amstrad CPC464 is starting to attract the attention of add-on manufacturers. One example is Computer Services who have announced that they are able to supply a lead to connect the computer to any Centronics compatible printer. The leads are available ex-stock from the company at 63 Quilp Drive, Chelmsford, Essex, CM1 4YD. The price is £14.87 fully inclusive.

Little surprise at Aries move

In what is described as a surprise move, Aries Computers have dropped the price of their Aries-B20 20K expansion board for the BBC micro by no less than 20%. The company have in the past few months been concerned that a patent which they hold in respect of the memory board is being infringed by other companies who, Aries claim, are producing cheap imitations. Aries contend that these pale imitations of their board will be the subject of punitive legal action. They also state that people will be in for a nasty shock when details of their patent application are published. Certainly many people are eagerly awaiting disclosure of these details as on the surface of it there seems little basis on which to establish a patent.

The fact that Aries can afford to shave 30% off the retail price of the board could be seen as an indication that the company are prepared to fight for a share of the memory add-on market on conventional commercial grounds rather than keeping the field to themselves by threatening legal action claiming infrigement of a patent that they seem reluctant to disclose.

The Christmas sales war

The contenders in the battle for home micro sales during the run up to the Christmas are starting to emerge from their summer hibernation. As with last year the sales effort will attack two groups of potential purchasers – those looking to upgrade their present computer and people making their first microcomputer purchase. These two groups can each be further subdivided according to the amount of cash the buyer wishes to spend on their computing habit.

Taking the first-time buyer group, the sales here are likely to be dominated by Commodore, Sinclair and Amstrad. Commodore's new C16 looks very strong in the sub £150 market although production of the machine is reported to have slipped back from the original target and it may not make its September launch date. Even if the C16 fails to make it to the shops in large numbers, Commodore still have two machines designed to appeal to this group. Despite persistent attempts to kill off the VIC20, the machine refuses to die and Commodore are still producing these machines to meet demand from dealers. The '64 is also a very strong representative in this area of the market.

The Sinclair Spectrum is another entry point machine that looks set to capture a significant slice of the Christmas present market. In view of the fact that some reports put the ex-factory cost of a Spectrum at about £15, it could stand a generous retail price reduction, this would maintain its edge over the C16 against which, at its present price, it looks rather weak.

Amstrad have a machine that will appeal both to the first time and upgrade markets. Initial response to the *E&CM* survey shows a number of Spectrum owners looking to upgrade to a CPC464 and Amsoft will have quite a range of software in the shops, a factor that makes the computer attractive to first timers. The only obstacle to the fortunes of the '464 will probably be the ability of Amstrad to get enough hardware into the shops.

The various MSX micros will be available in time for the Christmas rush although current indications are that these machines will not offer as attractive a price/performance ratio in comparison with most of the home grown micros. It remains to be seen whether or not the attractions of the idea of a standard microcomputer design will prove powerful enough to attract a large volume of sales.

At the upper end of the market, sales of the BBC micro are likely to remain strong although in most peoples' eyes sales of this machine have peaked and begun a steady downward spiral. If the Enterprise makes it to the shops in time this too will offer an attractive upgrade machine although lack of sufficient software support could mean that it will be next year before this machine makes any significant impact on the market.

Finally a list of also-rans,m machines that for one reason or other are unlikely to make much of an impact this year – Oric/Atmos, Electron, Dragon, Memotech, CBM Plus4, and the much talked about QL. In all these cases, and in particular, in the case of the QL, time could prove these predictions to be wildly out of tune with events. In any case this year is going to be one of the most interesting to date with so much good quality hardware chasing a limited number of micro users. Some companies at least will stand or fall by their performance over the next few months. The pattern of sales this year could well set the shape of the UK market during the next couple of years.

GARY EVANS

Spot the change at Dragon

When is a Dragon not a Dragon? The answer: when it's Touchmaster.

By now most of you will have heard that a company with the delightful name of Eurohard have purchased all assets of the defunct Dragon Data Limited and intend to start production of Dragon products in Spain some time later in the year.

Confirmation of this news reached us by way of a press release from the newly formed Touchmaster Limited. Touchmaster is an interesting company, it is based on the Kenfig Industrial Estate in Port Talbot, the same address from which Dragon Data traded and has on its staff Brian Moore and Richard Wadham, both of whom served time with Dragon Data. The company are a wholly owned subsidiary of Prutec, Prutec too had a stake in Dragon Data before the collapse earlier this year.

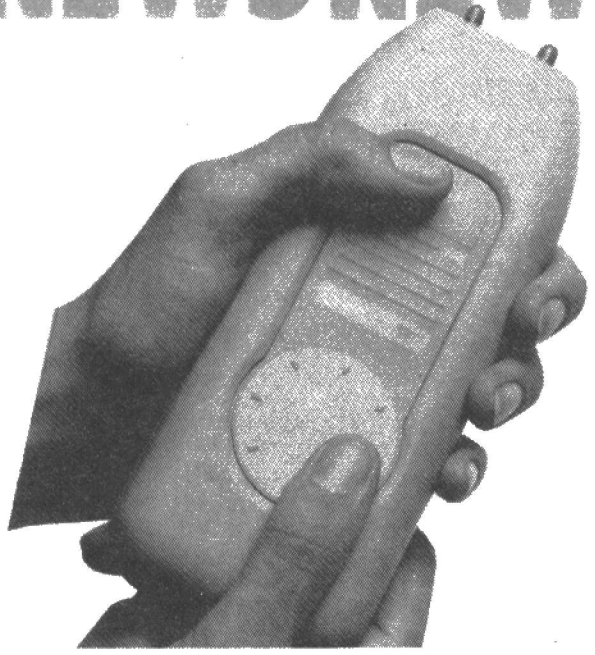
Touchmaster was set up to manufacture a pressure sensitive graphics tablet first seen on the GEC/Dragon stand at this year's CETEX exhibition. The press release states though that Touchmaster will also be offering Eurohard technical assistance with the production of existing Dragon products and with future technical developments, they are also apparently to be responsible for certain, unspecified aspects of customer support and software sales for Dragon products in UK territories.

All very confusing and when you hear the final ingredient to the tale is GEC, another name familiar to followers of Dragon tales, who will be responsible for sales and service of Dragon products you can see why we posed the question at the start of this item.

Full colour computer show

Whatever the future of the Dragon computer may turn out to be, there are still a considerable number of 32s and 64s in the hands of hobbyists. In view of this it is perhaps surprising that to date there has not been a show dedicated to these machines along the lines of the formula so successfully pioneered by the likes of the ZX and BBC exhibitions. Computer Marketplace (Exhibitions) Ltd. will plug this gap in the market later on this year. Their 6809 colour show will take place over the weekend of November 17th and 18th at the Royal Horticultural Hall in Westminster.

Any potential exhibitors requiring details of stands available should contact Tim Collins at Computer Market Place, 20 Orange Street, London, WC2 7ED. 01 930 1612.



RATs take control

Cheetah Marketing, the company recently acquired by Parc Electronics for a sum in excess of 1.4 million, have launched a new infra-red action joystick for the Spectrum. The RAT, which stands for Remote Action Transmitter, is designed to change the face of the more traditional conception of this peripheral.

The unit consists of two parts. The receiver, which plugs directly into the edge connector of the Spectrum and the transmitter. This is controlled by touch and has two buttons or "pads"; one of these is for firing and the second is to control direction and has a number of raised domes which are touch sensitive. The pad enables a very fast response and the infra red signals are transmitted by the light emitting diodes in the front of the transmitter over a fairly wide angle so that the unit does not necessarily have to be directly pointing at the screen.

No additional software is required for the unit and the company are hoping to make it compatible with other machines. The RAT will retail at £29.95 and is available from Cheetah Marketing, 24 Ray Street, London, EC1R 3DJ, Telephone number 01-833-4909.

More MSX news – details still scarce though

The companies behind the MSX computer standard are at last revealing more details as to their pricing policies and launch dates. The releases from the likes of Toshiba and Sanyo still abound with phrases such as 'the price is expected to be . . .', 'retail price will be around . . .' The signs are though that a typical MSX machine will cost £300 and that the first hardware will be in the shops in September. Toshiba are still favourites to win the race to the High Street but the others are unlikely to be far behind with Sanyo promising October delivery.

Tangerine 6809 add-on board is FLEX compatible

The Tangerine Computer can be transformed into a fully fledged FLEX system with the addition of Ralph Engineering's 6809 card. A Tangerine Computer equipped with the card does not suffer from the limitations of the Dragon computer when running under the same operating system. Installing the card converts the Tangerine into a fully functioning FLEX system and not just a terminal talking to a 6809 card.

Ralph Engineering, Forncett End, Norwich, Norfolk. 095389 420.

Compunet launch at PCW show

Compunet, Commodore's answer to the recently introduced CBM 64 Micronet service, has run into delays which have put back the launch of the facility until mid-September. The delays have been caused by difficulties associated with the host computer's software rather than any production problems concerning the modem that forms part of the package. It is hoped that the service will be ready in time for official launch at the PCW show and that people subscribing to the service at the show will be able to take a modem away with them and log on to the network as soon as they return home.

Compunet, incidentally, embodies a number of interesting features. The first of these is that the majority of the user terminal software is downloaded from the host computer at log-on – the firmware within the modem being the minimum required to establish the communications link between the two systems. This has a number of advantages, not least of which is that modifications and improvements to the service can be made by simply updating the host software. Another interesting aspect of the service is that downloaded software will be run time dongle protected – the dongle in this case being the users modem. Thus while Compunet software may be backed up to tape, to run it will require that the modem used to capture it will need to be present to run any program. As each modem has a unique ID embedded in its firmware this scheme offers a high degree of protection against unauthorised copying of any programs.

Out foxing the BABT

We hear that at least one multinational company with a stake in the home computer market is putting some experimental hardware through the BABT approval procedure. Apparently the firm are being rather coy when it comes to the full potential of the silicon contained within the prototypes and are taking a softly softly approach to the problems of getting equipment type approved.

This particular company also have a UK manufacturing facility and this makes the problems of obtaining approval for production machines a much simpler process than is the case with firms making use of overseas production facilities.

The prospect of a low cost computer, complete with built-in modem, now looks a real possibility within the very near future.

Statement

The publishers of *Electronics and Computing Monthly*, EMAP Business and Computer Publications would like to point out that the appearance of an advert within *E&CM* does not imply that the publishers condone or encourage any views or beliefs that may be portrayed by companies or organisations purchasing advertising space within the magazine. When confusion between paid for advertising and editorial matter is likely to occur, any paid for space will be clearly headed with the word ADVERTISEMENT.

The publisher.

PORT MASTER

Just the program you need in order to use a control interface with the BBC micro without the hastle of configuring the VIA's Data Direction register. Mike Williams explains.

More and more BBC computer owners are becoming interested in using their machine to interact with the outside world. They also want to explore the world of Robotics and Control as well as the sensing of temperatures, light levels, sounds and so on.

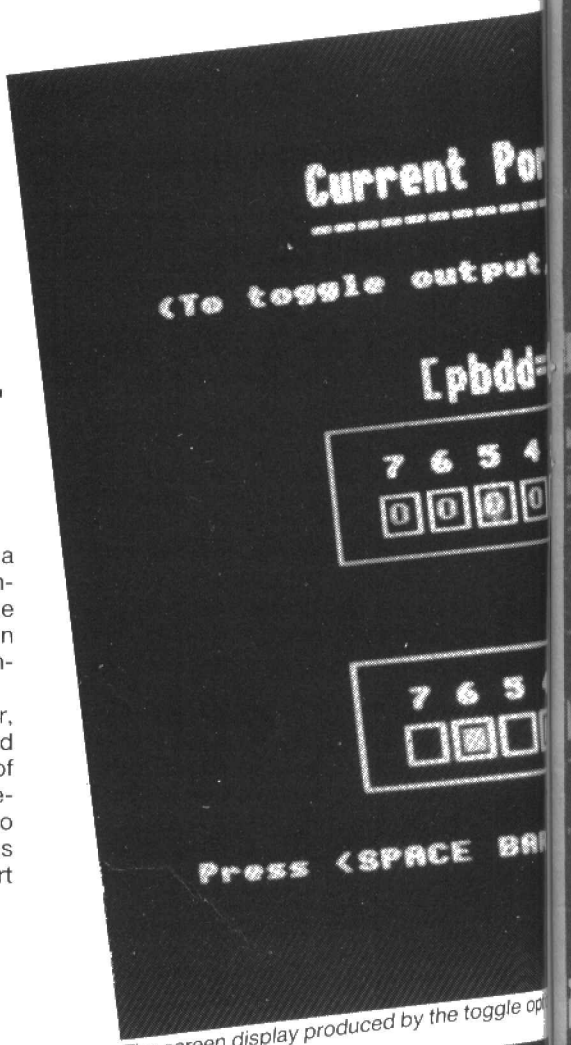
In Primary Schools very young children are using computers to switch lights on and off. A simple light sensor tells the computer when it is getting dark, and on go the lights. Traffic light systems can be simulated and some Logo type languages are coming along to control these systems without involving the children in the intricacies of BASIC syntax.

In order to connect up the computer with sensors or motors, some sort of interface board is usually built or purchased. This board normally plugs into the User Port and typically may allow four sensors to be read by the computer and four relays to be switched on and off. Many magazines have given instructions on how to build such a board, and they are also available commercially. (If you do connect straight to the User Port, without an interface board, then you run the risk of blowing up some expensive chips inside your Beeb!).

Having got the interface board however, there comes the problem of what is called 'configuring the Data Direction register' of the User Port VIA. Now if that is gobledegook and you have no wish to delve into hexadecimal code too deeply, then this is just the program you need. The Port Master allows you to:

set up the interface board;
decide the inputs and outputs;
switch the relays on and off;
monitor the inputs.

with only the bother of typing in the program.



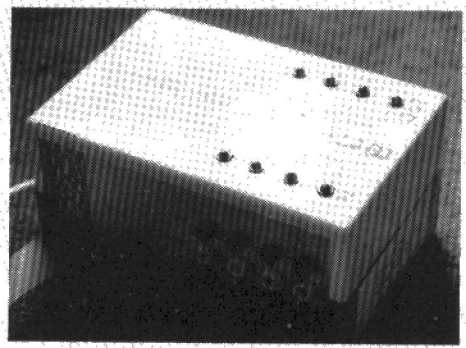
The screen display produced by the toggle op below.

PORT MASTER IN ACTION

To illustrate the operation of the Port Master software, we hooked up Pilot One's Interface (reviewed in the current issue of **Your Robot**) to the office BBC micro, and put the routine through its paces.

The interface requires that lines PB0, PB1, PB2, and PB3 are configured as output lines and that the other four lines (PB4, PB5, PB6 and PB7) are set up as inputs. The Port Master software was loaded and the I/O lines were set to the correct status after answering no to the initial prompt which asks whether or not the software was configured for correct operation. Setting the lines was straightforward and after this initial step the screen will display the four digit hex number that, when placed in line 410, would tailor the Port Master to the conditions required by Interface.

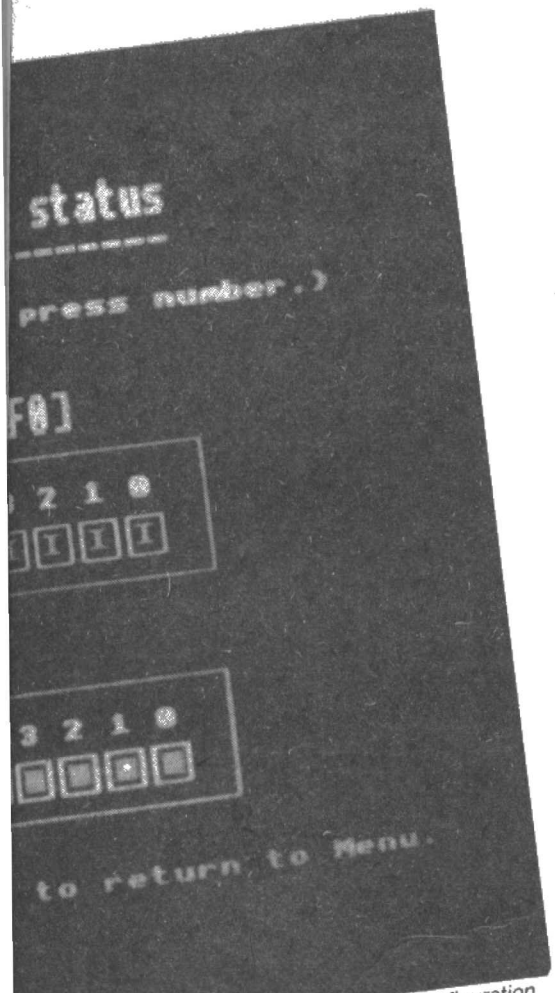
By selecting the toggle option from the Port Master menu, the four relays of Interface could be switched on and off – the switching action being confirmed both by the interface's mimic LEDs and the Port Master screen display.



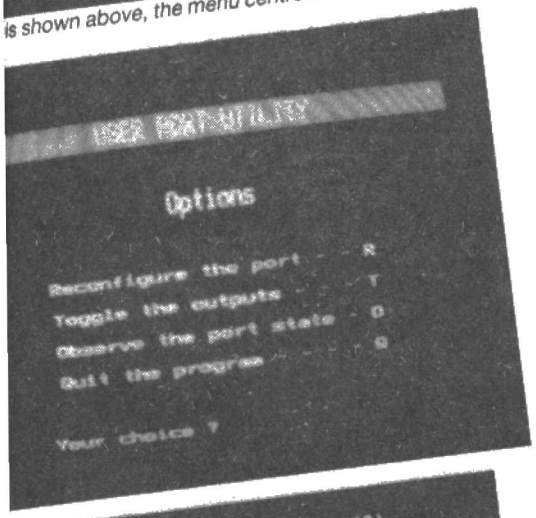
Pilot one's Interface.

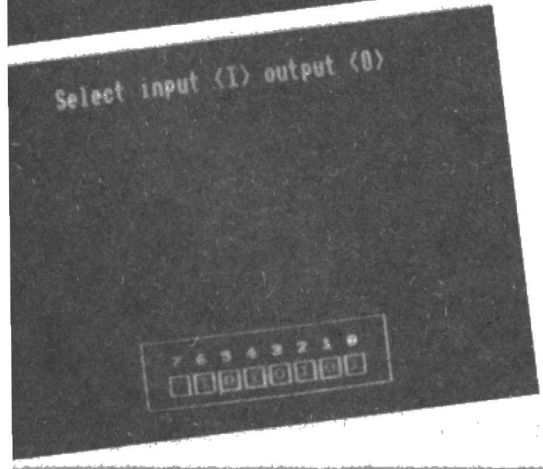
The input lines of Interface feature a schmitt trigger input which means that they can be used both with ordinary switches or with photodiodes, LDRs, thermistors etc. To confirm the operation of the input lines Interface was connected to a motor driven crane that was supplied by Pilot One as part of their review package. This featured an optically encoded disc that provides positional information as the crane's turret is rotated. Using a suitable combination of output lines, the turret was rotated and the pulsing 'LED' on one of the input lines displayed by Port Master confirmed that input monitoring was being performed by the software.

Port Master provides a quick and efficient means by which the input/output lines of any interface can be monitored. It should prove a useful addition to the software collection of anyone using their BBC micro to investigate the field of computer control.



is shown above, the menu centre and initial configuration






```
  100 REM     ********************
  110 REM     *  PORT UTILITY    *
  120 REM     *       by         *
  130 REM     *  M.E.Williams    *
  140 REM     ********************
  150
  160 ON ERROR GOTO 1350
  170
  180 PROCinit
  190 MODE1
  200 VDU23;8202;0;0;0;: REM remove cursor.
  210 PROCconfigure : REM set up the Data Direction Register.
  220
  230 REPEAT
  240   MODE7:PROCmenu
  250   MODE1:VDU23;8202;0;0;0;
  260   IF In$="R" THEN PROCconfigure
  270   IF In$="O" THEN out=FALSE
  280   IF In$="T" THEN out=TRUE
  290   IF In$<>"Q" THEN PROCread_port
  300   UNTIL In$="Q":MODE7:END
  310
  320
  330 DEF PROCinit
  340 @%=1:*TV0,1
  350 :
  360 VDU23,225,127,127,127,127,127,127,127,127:REM led
  370 portB=&FE60  : REM PortB port address.
  380 pbdd=&FE62   : REM PortB Data Direction register.
  390 DIM block% 9 : REM used by double sized letter procedure.
  400 out=FALSE
  410 ENDPROC
  420
  430 DEF PROCconfigure:LOCAL N%,J%
  440 CLS
  450 PROCdouble(" Are you using the interface board?")
  460 IF FNy_n THEN ?pbdd=&F0:CLS:ENDPROC
  470 REM The default interface board has 0123 Inputs,4567 Outputs.
  480 REPEAT
  490   CLS:PROCdouble("     Select input <I> output <O>")
  500   PROCleds(25)
  510   REM Now get the appropriate number,N%,for Data Direction
register.
  520   N%=0
  530   COLOUR1
  540   FORJ%=8 TO 1 STEP-1:PRINTTAB(2*J%,3);"?"
  550     In$=GET$
  560     IFIn$="O" THEN N%=N%+2^(8-J%):PRINTTAB(2*J%,3);"O" ELSE IF
In$="I" PRINTTAB(2*J%,3);"I" ELSE GOTO 550
  570     NEXT
  580   COLOUR3
  590   VDU26 : REM Reset window.
  600   PRINTTAB(14,29);
  610   PROCdouble("Is this OK ? ")
  620   UNTIL FNy_n
  630 ?pbdd=N%
  640 ENDPROC
  650
  660 DEF PROCleds(y_pos) : REM print boxes for 'leds' and frame.
  670 FOR J%=1 TO 8
  680   PROCbox(10+2*J%,y_pos,1,1,TRUE)
  690   NEXT
  700 PROCbox(10,y_pos-3,19,5,TRUE)
  710 FORJ%=0 TO 7
  720   PRINTTAB(2+2*J%,1)7-J%
  730   NEXT
  740 ENDPROC
  750
  760 DEF PROCread_port
  770 CLS
  780 PROCdouble("          Current Port status")
  790 PRINT:PRINT
  800 PROCdouble("          ------------------")
  810 IF out THEN PROCcenter("(To toggle output, press number.)",5)
  820 COLOUR2
  830 PROCcenter("Press <SPACE BAR> to return to Menu.",30)
  840 PRINTTAB(14,9);
  850 PROCdouble("(pbdd=&"+STR$~(?pbdd)+")")
  860 GCOL0,2
  870 PROCleds(15)
  880 COLOUR1
  890 status%=?pbdd
  900 :
  910 REM check input and output lines and print on screeen.
  920 FORJ%=1 TO 8
  930   IF (status% AND 2^(J%-1)) THEN PRINTTAB(18-2*J%,3);"O" ELSE
PRINTTAB(18-2*J%,3);"I"
  940   NEXT
  950 COLOUR3
  960 GCOL0,3
  970 PROCleds(25)
  980 :
  990 REM Indicate state of port using red 'leds'.
```

When the program is run, the user is asked if he/she is using the interface board. This refers to the board which will be attached to the port, and assumes that the Port Master has been run before and is now tailored to the board. If the interface board is being used for the first time then enter 'no' here. You will then be asked to select which of the port terminals are to be used as inputs and which as outputs. The instructions with your board should make the choices clear to you.

The options shown in the menu are now presented.

Reconfiguring the port

Means changing the arrangement of inputs and outputs.

Toggle the outputs

Produces the current port status display. The top box shows which of the 8 terminals in the User Port are outputs and which are inputs. The lower box shows the actual state of the terminals – either at 0V or at 5V (shown by the red 'led').

The number following the pbdd= in the yellow brackets is the number which should be in line 460 if the program is to be customised for your own interface board. For those versed in the mysteries of the User Port (lucidly explained by Paul Beverley in the June edition of *Electronics and Computing Monthly*) the number describes the contents of the Data Direction register.

The outputs can be changed by pressing the corresponding number key. Of course pressing the key of an input has no effect.

Observing the port state

This is the same as the previous ption but without the possibility of changing any outputs.

The program is liberally sprinkled with REMs and is fairly well structured. For those interested I give a brief description of the procedures used.

PROCinit

Sets up some variables.

PROCmenu

Produces the menu display. It uses some nice Mode 7 procedures at lines 1500 onwards.

PROCconfigure

Arranges the contents of the Data Direction register. In other words it tells the User Port which of its lines are inputs and which outputs. The default value I have given is &F0. This gives four inputs and four outputs. If you are reconfiguring the port (or you answered 'no' at the start of the program) then the loop from lines 480 to 620 calculates the appropriate number for you. To understand how this works, go back and reread Paul Beverley and all should be clear.

PROCleds

Draws the boxes which contain the red 'leds' and the I or O letters. It calls PROCbox which in turn calls PROCwindow. These are super procedures which draw frames around text windows of any size. I'm sure you will find them useful.

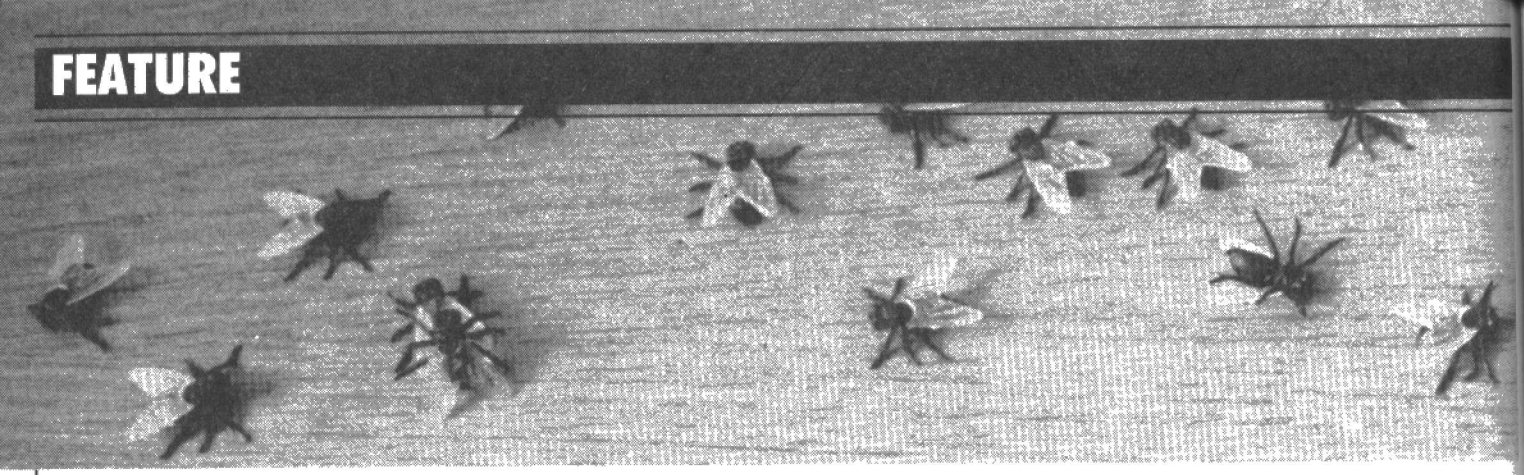
```
1000 REM The program spends most of its time in this loop.
1010 REPEAT
1020   status%=?portB :REM read the current state of the port.
1030   FOR J%=1 TO 8
1040     IF (status% AND 2^(J%-1)) THEN COLOUR1 ELSE COLOUR0
1050     PRINTTAB(18-2*J%,3);CHR$225
1060     NEXT
1070   IF out THEN J%=INSTR(" 01234567",INKEY$(0)):IF J%>1 THEN PROCout
1080   UNTIL INKEY(-99)
1090 ENDPROC
1100 :
1110 DEF PROCout: REM toggle one of the outputs.
1120 J%=J%-2:IF(?pbdd AND 2^J%) <> 2^J% ENDPROC
1130 SOUND1,-10,53,2
1140 status%=?portB
1150 IF(status% AND 2^J%) THEN status%=status%-2^J% ELSE
status%=status%+2^J%
1160 ?portB=status%
1170 ENDPROC
1180 :
1190 DEF PROCmenu
1200 PROCheader("USER PORT UTILITY")
1210  PRINT'':PROClarge("Options")
1220 PROCcenter("Reconfigure the port - - R",10)
1230 PROCcenter("Toggle the outputs - - - T",12)
1240 PROCcenter("Observe the port state - O",14)
1250 PROCcenter("Quit the program - - - - Q",16)
1260 PRINT''''"      Your choice ?   ";:PROCinput
1270 CLS:ENDPROC
1280 :
1290 DEFPROCinput
1300 REPEAT
1310   In$=GET$
1320   UNTIL INSTR("RTOQ",In$)
1330 ENDPROC
1340 :
1350 MODE7
1360 PRINT"I regret to inform you, oh master,"'"of an error:"
1370 REPORT:PRINT" in line ";ERL
1380 END
1390 DEFPROCwindow(SX,SY,W,H,F,B):REM
start,width,height,foregnd,backgnd
1400 VDU28,SX,SY+H-1,SX+W-1,SY:COLOURF:COLOURB+128:ENDPROC
1410
1420 DEFPROCbox(x%,y%,w,h,border):REM box & window origin is top left
1430 LOCALw%,h%
1440 IF border THEN PROCwindow(x%,y%,w,h,7,0)
1450 w%=w*32+16:h%=h*32+20:y%=31-y%-h
1460 MOVEx%*32-8,y%*32+20
1470 PLOT1,0,h%:PLOT1,w%,0:PLOT1,0,-h%:PLOT1,-w%,0
1480 ENDPROC
1490
1500 REM***** MODE 7 procedures *****
1510 DEFPROCcenter(A$,vpos)PRINTTAB((19-LENA$DIV2),vpos);A$:ENDPROC
1520 DEFPROCheader(A$)
VDU30:FORI=0TO1:PRINTCHR$&81;CHR$&9D;CHR$&83;CHR$&8D;:PROCcenter(A$,VPOS
):NEXT:ENDPROC
1530 DEFPROClarge(A$)FORI=0TO1:PROCcenter(CHR$&8D+A$,VPOS):NEXT:ENDPROC
1540
1550 DEFFNy_n:LOCALI%:REPEAT:I%=INSTR("Y,y,N,n",GET$):UNTILI%
1560 IFI%<3 =TRUE ELSE=FALSE
1570
1580 DEFPROCdouble(A$):LOCALI%
1590 FORI%=1TOLEN(A$)
1600   PROCchar(ASC(MID$(A$,I%,1)),224)
1610   NEXTI%:ENDPROC
1620 :
1630 DEFPROCchar(C%,char%)
1640 LOCALA%,X%,Y%,J%,I%
1650 ?block%=C%:A%=10:X%=block%:Y%=block% DIV256:CALL&FFF1
1660 FORJ%=0TO1:VDU23,char%
1670   FORI%=2TO9:VDUblock%?(J%*4+I%DIV2):NEXT
1680   VDUchar%,10,8:NEXT
1690 VDU11,11,9:ENDPROC
1700
1710 DEFPROCwait(t%):TIME=0:REPEAT UNTIL TIME>t%:ENDPROC
```

PROCread_port

This produces the port status display. The instruction concerning output toggling only appears if option T has been selected (in which case variable 'out' is TRUE).

The program probably spends most of its time in this procedure and in particular in the loop at lines 1010 to 1080. The faster the program gets through this loop the more immediately it will respond to any change. For clarity I have spread things out, but for speed you could remove the REM and squash some of the lines together. But BBC BASIC is so impressively fast that I doubt if you will need to.

If you try to toggle an output then PROCout first checks with the data direction register to see if you are trying to toggle an input. Such impudence takes you out of the procedure. Otherwise you get a confirming beep and lines 1140 to 1160 effect the change.



BASIC BUG HUNTING

Mike James shows that, by adopting a systematic approach to program debugging, the task can be made very much easier.

BASIC is an excellent programming language; it is easy to use and well suited to the range of applications for which home micros are used. But no matter how excellent the language and how experienced the programmer, "bugs" – program errors and misbehaviour – are a fact of life.

Every programmer has to learn to deal with bugs in one way or another, but it is surprising how often debugging is carried out in a haphazard and non-systematic way. No-one would dispute that programming is, fundamentally, a logical activity but when it comes to bugs everything seems to go haywire along with the program. Perfectly rational programmers seem to believe that bugs are best dealt with by inspiration and divine guidance – in practice the only thing that this approach achieves is a great waste of time.

It is often difficult to see how finding a particular bug in a program running on a particular machine can possibly be generalised and a lesson learnt that will help with finding future and very different bugs. To a certain extent this is a real difficulty, but there are general principles that lie behind debugging and there are specific types of fault that tend to occur in any given language.

The first part of this article deals with general principles of debugging a program and the second part discusses some of the most often encountered mistakes in BASIC. Notice that this list of common errors is by no means complete because there is a personality factor which causes some programmers to be prone to certain errors that other programmers never seem to make.

Just looking

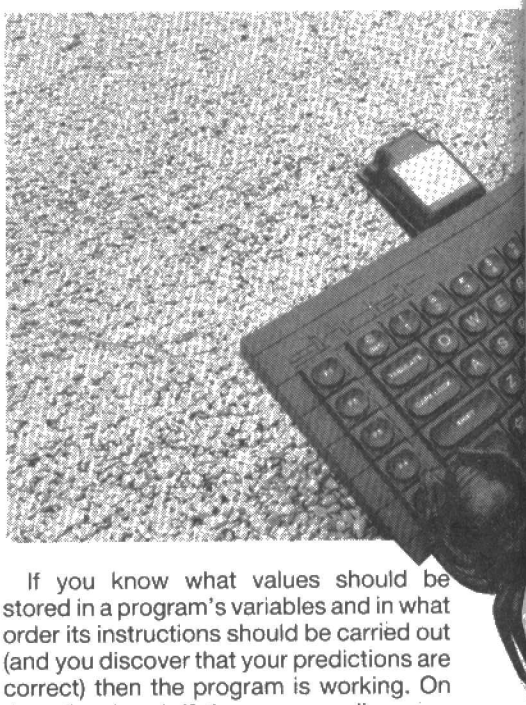
The most common approach to finding bugs is the "just looking" method. What is surprising is that a simple examination of the listing of the program does in fact solve most of the bugs that crop up in the early stages of program development – this is simply a consequence of such bugs being very simple and very obvious. The danger is that this early and powerful success of the "just looking" method results in it being used in situations where it is totally ineffective.

If you have detected a bug by way of the program misbehaving or by getting an error message, then by all means examine the program, using any knowledge that you may already have to narrow down the area that you look at, BUT if this doesn't reveal the source of the error in a few minutes then move on to the logical debugging methods described below.

Debugging – Rule One

If you fail to find the problem after a few minutes DO NOT continue simply to stare and ponder over the listing. There is only a very limited amount of information a listing can give you about how the program is working – after all it is a "static" representation of a "dynamic" program. Put simply – to effectively debug a program you need as much information about how your program is working as you can easily obtain. In this sense debugging is an active rather than a passive examination of the program. To debug quickly and efficiently you have to hunt and trap the bug rather than just wait for it to show itself as the result of inspiration.

The need for information about how a program is running is obvious, but exactly what information is required? Although you may think that the answer to this question would be difficult it is quite straightforward and precise. There are only two components to the way a program works; the values stored in its variables and the order in which its instructions are carried out, and these two components are also responsible for the way a program fails to work!

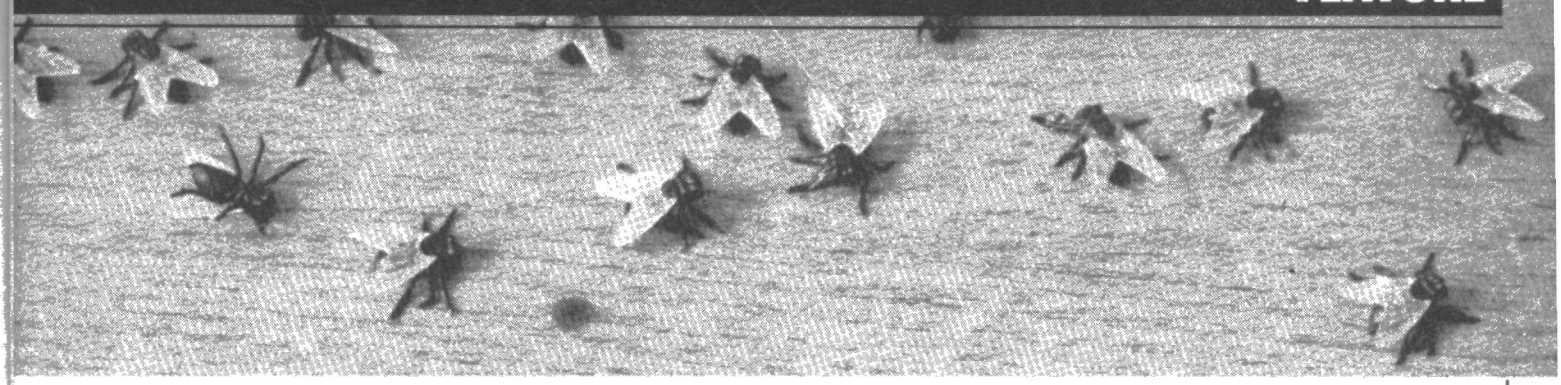


If you know what values should be stored in a program's variables and in what order its instructions should be carried out (and you discover that your predictions are correct) then the program is working. On the other hand, if the program disagrees with your predictions then it is clearly not working as intended – that is, it contains a bug. The first place that the program disagrees with your prediction is the most likely place for the bug to be situated.

"general principles lie behind debugging and specific faults that occur in any language".

Putting this theory into practice is quite simple. If your program misbehaves and you cannot immediately see what the problem is, then you have to begin the process of predicting the values stored in each of its variables and the order that its statements will be carried out, then compare these predictions with reality.

To find out what is really stored in a variable is a matter of temporarily adding a PRINT statement to the program to print out the current values that each variable holds. In practice it is usually fairly easy to avoid the information overload produced



by printing the values stored in every variable; this is done by picking a small number of variables that are obviously important. The order that statements are executed in can also be discovered by adding temporary PRINT statements which display the next or their own line number.

Once again in practice the information overload produced by printing a message

just before each line is executed can be avoided by just adding PRINT statements to show which way the program went following each IF statement. Only where an IF statement presents a choice of statements to be executed is there any doubt about the order in which they are carried out – although see the "fall through" error described below.

The art in using logical debugging is to get exactly the amount of information you require and no more. For example, if a program presents the user with a selection from a menu but fails to jump to the correct choice, then clearly you are interested in finding out the value of the variable used within the program to record the user's selection and no others. If the value in this variable is correct, then you must examine the exact order that statements within the menu section of the program are carried out. Are they ever executed? Does the program always follow the same route? Does the program go to the correct option and then jump somewhere else . . .? Notice that it is important to predict what you expect the program to do BEFORE you examine what it does. The reason for this is that most programmers believe in their program to the extent that they are too willing to assume that it is correct!

The second rule of debugging is to predict the values stored in each variable and the order of execution of each statement, and then check that this IS how the program works.

As already described, in practice things are not quite so bad in that you do not have to predict and examine all of the variables and all of the statements if you have some idea where the bug is in the program.

This is all that there is to logical debugging – easy isn't it? What is surprising is how keeping the principle of logical debugging in mind can make the task of finding bugs easy. Even though this principle of debugging can and should be applied to programs written in any language, it can save a great deal of time if you are aware of the most probable types of errors to which each language gives rise. As already mentioned, BASIC is a good language but it has its faults, and the time has come to examine them – starting with the most serious.

The trouble with line numbers

A whole family of errors and problems arise either directly or indirectly from BASIC's dependence on line numbers. In nearly all versions of BASIC (there are one or two exceptions) each line in a program must have a line number. This is so much part of the BASIC language that BASIC programmers rarely stop to question if so many line numbers are really necessary. BASIC line numbers serve two purposes – as convenient "markers" for editing program lines and as 'labels' that are used by GOTO and GOSUB commands. It is clear that, as far as editing is concerned, every line must have a line number for the method to work, but for the use of GOTO and GOSUB the only lines that NEED line numbers are those that are actually referred to in GOTO or GOSUB commands.

"It is important to predict what you expect the program to do BEFORE you examine what it does".

There are many reasons why the dual function served by line numbers causes a problem in the easy production of clear and bug-free programs.

The line misplacement problem

The fact that every line has to have a line number which determines its position within the program is a mechanism that is very sensitive to typing errors. You only have to mistype a single digit and, not only does a line appear to be missing at one position, an extra line appears elsewhere, possibly in part of a program that was finished and debugged some time before. In this sense, typing a single digit incorrectly destroys the overall correctness of the program and introduces a very difficult bug, lurking in a part of the program that you have tested and is therefore almost above suspicion!

If a program worked until you typed in a few extra lines, then suddenly goes berserk, it is worth checking that all the lines you entered are present where you expect them to be. If you type in a line and it goes missing then don't simply type the line in again – find out where it has gone and get rid of it. Also, the line misplacement problem is a good reason for not typing in many lines of a program at a time. Type in a small block of lines and then check that they are all present and correct.

Fixed destinations

When a line number is used in a GOTO or GOSUB statement then it is an advantage if it remains unchanged throughout the life of the program. The reason for this is that the line number of the destination of a GOTO or GOSUB serves to 'name' a part of the program that performs a particular task. If this 'name' changes during the development of a program then it is possible that the programmer will incorrectly use the 'old' value instead of the correct new value when re-using that part of the program.

For example, if during the initial implementation of a program, subroutine 1000 prints an opening message then you are likely to use GOSUB 1000 later on to print the same message. If, however, in the intervening time it has been necessary to renumber the program so that extra lines can be inserted, line 1000 may be assigned a value, say 1370.

The forward jump problem

When writing a program, it is easy to use a GOTO statement to transfer control back to a point earlier in the program. The reason for this is that it is usually the earlier part of the program which is already written

and it is very easy to find the line number of the point to which you want to return. This should be compared to the situation when using the GOTO to transfer control forward to a line later in the program. In this case the chances are that the line to which you are trying to transfer control has not yet been written. The line number that will eventually be assigned is difficult to work out, as it depends on the number of statements that will be written and the line number increment in use.

The best solution to the forward jump problem is not to make a guess at the line number, but to use a symbol such as "*" to indicate that you have to return and fill in the missing number. If you guess the line number, it is possible that you will get it wrong and this will go undetected because, even though the line number you used was incorrect, the GOTO is still valid BASIC. If you use "*" to mark incomplete GOTO statements it is impossible to forget them as they will cause the program to crash if not removed!

It is possible to imagine a version of BBC BASIC that doesn't use line numbers for editing. Program lines could be entered, deleted, listed and modified by use of the cursor keys instead. This would indeed be a great improvement in that the confusion and difficulties that result from the dual use of line numbers would be completely eliminated. Failing this, the best course of action would be to try to avoid the use of line numbers within GOTO and GOSUB by making GOTO redundant and by replacing the GOSUB with a PROC call as in BBC/Acorn or QL Super BASIC.

In and out of FOR loops

FOR loops seem to cause rather more trouble than they should, perhaps because their purpose is not always made clear in introductions to BASIC. A FOR loop is intended to be used whenever a group of statements needs to be repeated a given number of times. This is simple enough, but it often becomes confused with the closely related 'conditional loop' – that is repeating a group of statements until some logical condition is satisfied. The result of this confusion is the tendency for BASIC programmers to jump out of FOR loops before they are properly complete. For example, consider the problem of scanning a string to find the first occurrence of a particular letter or, failing that, the end of the string. This is most often programmed something like:

```
1000  FOR I=1 TO LEN(S$)
1010  IF MID$(S$,I,1)=T$ THEN GOTO
      1030
1020  NEXT I
1030  rest of the program
```

where S$ is the string to be searched for the character in T$$. In many versions of BASIC this will result in the program crashing; even in those versions of BASIC where this is allowed it is still poor programming style. The point about FOR loops is that you should be able to work out how many times they will repeat by looking at the FOR statement itself (line 1000 in this case) and not have to worry about statements within the loop. The jump out of the FOR loop at line 1010 is not only confusing but leaves the loop unfinished and most versions of BASIC cannot deal with an unlimited number of unfinished FOR loops. The premature exit from a FOR loop will eventually crash the program.

A solution to this program that is often used, and even advocated as a sign of advanced programming technique, is to set the index variable equal to the final value and then jump to the end of the FOR loop. In the above example this would mean changing line 1010 to:

```
1010  IF MID$(S$,I,1)=T$ THEN
      L=I:I=LEN(S$)
```

L is used to save the position of the character in the string and then I is set to a value that will cause the FOR loop to end naturally. Unfortunately, although this does stop the program from crashing, it is still not good programming practice – it results in programs that are "unstable" in the sense that a small change in the way that the FOR loop is implemented, or in the program itself, can create a bug. The rule is that you should never jump out of a FOR loop and you should try to avoid bringing a FOR loop to an early end. It has to be admitted that sometimes a particular dialect of BASIC is so inefficient that you do have to resort to tricks of this sort to make a loop run fast enough – but then good programming is often about making a trade-off between efficiency and good style.

The fall through problem

The fall through problem is something that every programmer has experienced in BASIC. At its simplest it is caused by forgetting to put a GOTO, END or RETURN at the correct place in a block of statements. For example, if you forget to complete the main program with END then after you have completed a RUN, instead of halting, the program will "fall through" into the first subroutine following the main program. As this subroutine is often the first that the main program calls, it can appear that something very strange has happened and the program is trying to run itself from the beginning again!

In the same way, not ending a subroutine with a RETURN causes the next subroutine to be entered without being called from the main program. This also usually results in a subroutine being mysteriously called twice!

The final type of fall through problem often arises because of the tendency to forget to skip over a block of statements that you do not want to be executed. For example, if you want to execute "list 1" of statements if A is positive and 'list 2' of statements if A is negative then the correct way is to use something like:

```
1000  IF A>=0 THEN GOTO 1500
      .     .     .
      list 2
1490  GOTO 1600
1500  .     .     .
      list 1
1600  rest of BASIC program
```

where line 1000 skips "list 2" if A>=0 and 1490 skips "list 1" if A<0. The most common error is to forget to enter line 1490 and so "list 1" is executed whenever "list 2" is – in other words "list 2" falls through to "list 1" without line 1490. There are plenty of other examples of the fall through bug but they can all be found by tracing the flow of control. Looking out for them, however, can save some time.

Expressions and conditions

Expressions and logical conditions form the main way that anything useful gets done in almost all computer languages. It is surprising to say that; even after many years of programming; it is still possible to be confused by the slight differences between the BASIC form of writing arithmetic and logic and the English way. For example, if you try to convert the formulae:

$$\frac{1}{y(1+X)}$$

into BASIC be careful not to write:

```
1/Y*1+X
```

or even

```
1/Y*(1+X)
```

the correct form is (of course)

```
1/Y/(1+X)
```

but the double use of the divide symbol makes it look odd. Logical expressions can also be confusing. For example, if you want to check that the answer to a question was either "YES" or "NO", then you might use:

```
IF A$<>"YES" AND A$<>"NO" THEN
GOTO error
```

Many programmers consistently write OR in place of the AND in this logical expression. The moral of both these tales is not so much that you should look out for these particular examples; it is much more that you should never assume that an expression of any sort actually calculates what you think it does! In other words, the "checking values" part of logical debugging is something to take very seriously – even when you are convinced that you know the answer.

EPROM SIMULATOR – AID TO M/C DEVELOPMENT

Richard Sargent and Robert Harvey describe an EPROM simulator that can dramatically reduce the time taken to develop machine code programs. The project was designed to offer maximum flexibility combined with ease of use.

The EPROM simulator is a small pcb card which contains ten standard logic and memory chips and which behaves as though it were a 4K Eraseable ROM. It can be attached to a single computer or it may link one computer to another system. In either case it acts as a convenient buffer between the controlling system (CONTROLLER) and the system under development (RECEIVER) (see **Figure 1**).

Essential test equipment

Suppose a stand-alone logic unit is being designed to control the total environmental conditions of a greenhouse. The unit will consist largely of input and output devices and naturally a CPU and ROM will handle the decision-taking and control the whole show. Generally, the minimum amount of equipment which would be needed to develop such a system would be:

1 A computer with an Editor/Assembler for writing the greenhouse software. (A cross-assembler would be needed if the computer's CPU has a different instruction set to that of the greenhouse CPU).

2 An EPROM programmer to blow the software into one or more EPROMs.

3 An ultra-violet light-box for erasing all the EPROMs that didn't work first time.

The EPROM simulator helps to develop the new system in two very important ways. The ultra-violet hardware may be dispensed with altogether, and the time taken to develop and blow a bug-free ROM can be drastically reduced. When you consider that it may take about ten minutes to set up and blow a ROM and thirty minutes to erase it again, and that the system may not work until version twelve of the software has been evaluated, it is likely that human patience and the physical strength of the EPROM will begin to wear thin!

Interfacing the simulator

The unit can simulate 2708, 2716 and 2732 EPROMs and connects to the RECEIVER system by ribbon cable into the ROM socket. The CONTROLLER loads software into the unit by means of a standard Centronics parallel interface.

The software under development can be downloaded from the controlling computer to the receiving system in a matter of seconds. This allows small test routines to be devised and placed quickly into the new system for part-testing of various circuit elements. Patches of code can be easily tagged onto the software under development, or parts of the coding may be overwritten. All this can be done easily and quickly and with no swopping of test leads or unplugging of ICs.

The hardware

The circuit is based around 4K of CMOS RAM, the contents of which simulate the EPROM of the system under evaluation. Two HM6116P-3 devices were used in the prototype. These are 2048 X 8-bit high-speed static RAM chips, capable of retaining their data on a standby voltage of 3.0V whilst drawing a 100μW current. They are made by Hitachi and versions with an access time as fast as 200ns may be obtained. The pin designation is compatible with the standard 2K ROMs, although that is of no significance in this particular circuit design. The pins are TTL compatible.

The block diagram (**Figure 2**) shows the principle of operation, with the RAM occupying the heart of the unit and data entering from the controlling computer on the left and passing out to the development system on the right. Data flow is controlled by SW1 and timing is controlled by the STB signal. Moving on to the circuit diagram (**Figure 3**) SW1 marks a convenient starting point for explanation. By controlling the chip-enable pin of IC3 and the address information through logic switches IC5, IC6 & IC7, the EPROM simulator can be put into one of two modes: Receive software (LOAD) or Simulate Eprom (SROM).

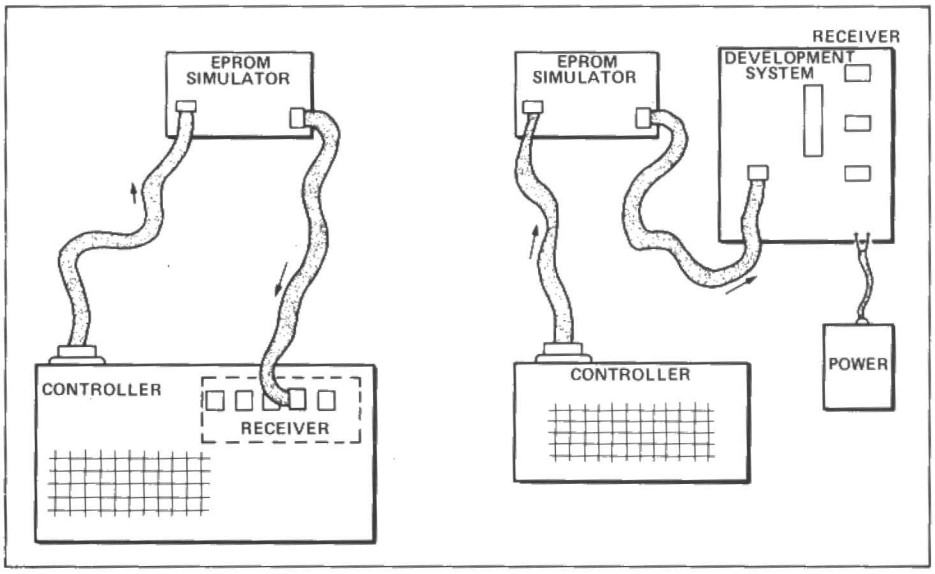


Figure 1. The relationship between the various items in a typical application.

The parallel interface provides the data sequentially one byte at a time, and the address of each data byte is generated by the CMOS counter 4040 (IC8). Three 74LS157 multiplexers (ICs5,6,7) take address information either from the counter (LOAD mode) or from the address bus of the EPROM socket (SROM mode). When SW1 is put into LOAD mode the output of IC9A becomes low, allowing the centronics data to pass through buffer IC3 onto the RAM data bus. IC10C illuminates LED D7 which indicates the LOAD mode. The multiplexers are correctly set, and the differentiator network R3/C1/D1 provides a spike pulse to the re-set pin of the address counter. Notice that the write enable (pin 21) of the RAM (ICs 1 & 2) is only enabled when the strobe pulse from the centronics interface occurs. This strobe must be a low going pulse and the parallel data must be valid for the duration of the pulse. The duration of the pulse should be a little longer than the access time of the RAM, so 300nSec should prove to be sufficient. The strobe is inverted by IC9C and increments the address counter IC8. On the count of 2048 the first RAM chip (IC1) is completely loaded with data and for the next 2047 bytes the high on pin 1 or IC8 will close down IC1 and allow the second RAM chip (IC2) to be loaded with the final 2K of data. Software should ensure that the transfer of data stops after 4K bytes have been transmitted, but if there is any "over-run", perhaps caused by the occasional spurious pulse then pin 1 of IC8 will go low and this condition can be reported on the BUSY line of the centronics interface.

The EPROM simulator connects to the recipient ROM socket via a 24-way DIL header. The EPROM's OE and CE signals are brought to IC10B and when low they enable the uni-directional tri-state buffer IC4. There can never by any contention on the simulator bus between signals from the controlling and receiving systems.

Three rail EPROMs

The EPROM simulator caters for all types of 24-pin EPROM. The 2708 and 2716 types are seldom advertised these days, but it is likely that a great many "second-hand" devices will be available on the hobby market for some years and you might find yourself using them. The +12 and —5 voltages from the EPROM socket would normally ruin the TTL in the simulator, so the protection network of D5, D6, R6 and R7 has been included on the address lines 21 and 19. The network is transparent to the normal high/low transitions of 5volt TTL, but when a three-rail device is connected pin 19 (EPROM socket) is read as TTL high and pin 21 is read as TTL low by the simulator. This leads to a rather strange mapping of the three-rail EPROMs onto the 4K RAM of the simulator because the extra voltages on the one type of EPROM are on the same pins as the address information of the larger capacity EPROMs, and 2708 data needs to be stored in the second 1K block of IC1.

Controlling the card

The prototype EPROM simulator has worked well from a Centronics interface where BUSY is tied LOW and ACK is tied to STB. 4K blocks of data move without error from the controlling computer to the simulator card. The purpose of LK1, LK2 and LK3 is to make the card as flexible as possible so that it can work from controlling computers where the information relating to the Centronics interface might be vague or missing altogether. A different approach would be to send the developed software out through the Centronics port using a purpose-written machine-code program with the BUSY (or the ACK) line testing for an over-run condition as previously explained.

For computers which have no parallel output port IC10D and LK3 have been provided to allow experimentation using the PORTSEL (port select) line. Remember though that IC3 is a buffer rather than a latch, and that any signal fed to PORTSEL should only go low at the controlling CPU's WRITE time and at MREQ (or IORQ) time. The CPU WRITE strobe (NWRS) may be tried as the strobe signal to the Simulator board, but close attention would have to be given to the timing waveforms of the CPU. It is probably easier in the long run to make up a 9-bit output port to interface between the controlling computer and the simulator card.

Normally the EPROM simulator takes its power from the supply pins of the EPROM socket, but a Nickel Cadmium rechargeable battery is used to power the RAM when the receiving system is powered down. This is essential if the receiving system uses the "switch of the power" method of RESET. (*E&CM* readers wouldn't do that surely!) The NiCad is trickle-charged via R5 and D3 when power is applied to the circuit, and care should be taken to ensure that the NiCad is sufficiently charged before pulling out all the plugs. The enable switch SW2 should be closed prior to any intended battery operation. There is no reason why the power supply should not be taken from the controlling computer, and link LK2 allows for this. Care should be exercised so that the EPROM simulator does not take its power from both systems simultaneously, and that the power supply driving the simulator is not overloaded by the extra demands of the simulator ICs and NiCad charger.

Construction and testing

Construction is reasonably straightforward if the printed circuit board is used. The CMOS ICs must be socketed and should be placed into the board last, observing the usual handling precautions of earthing

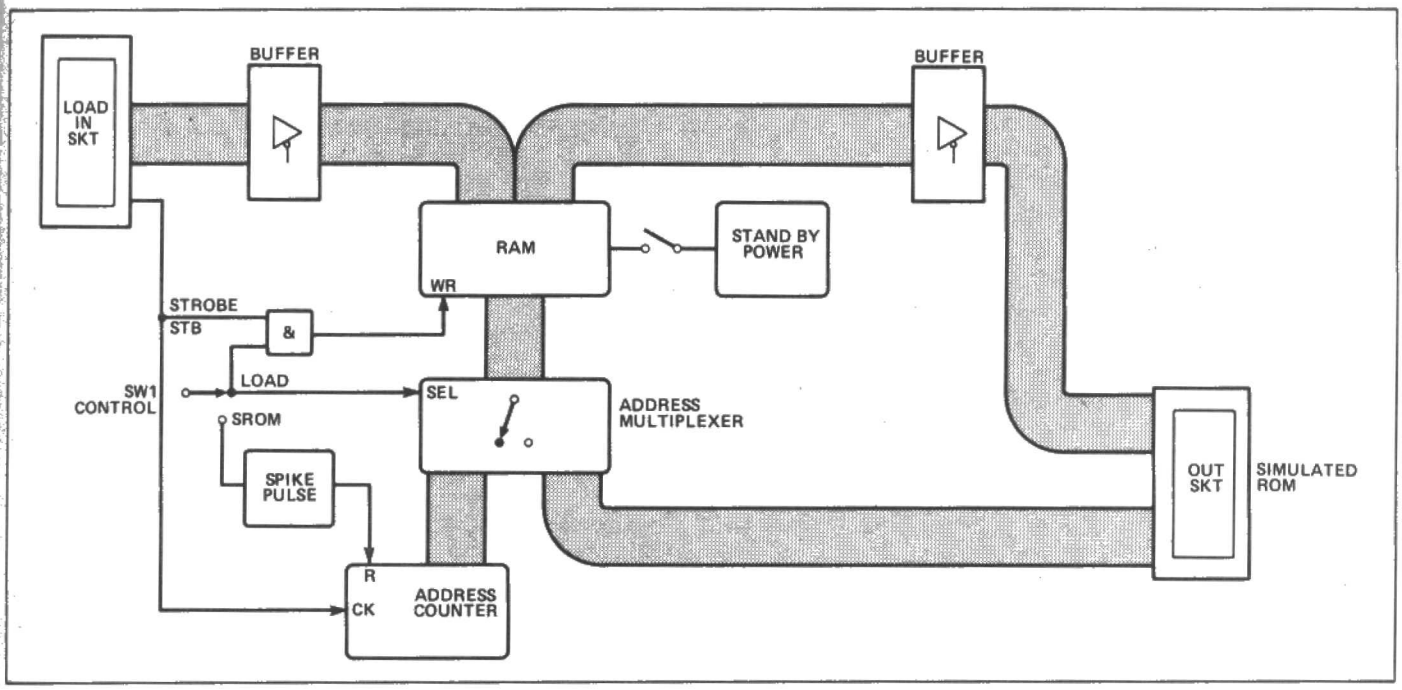


Figure 2. Block diagram of the system showing the RAM is the centre of the design.

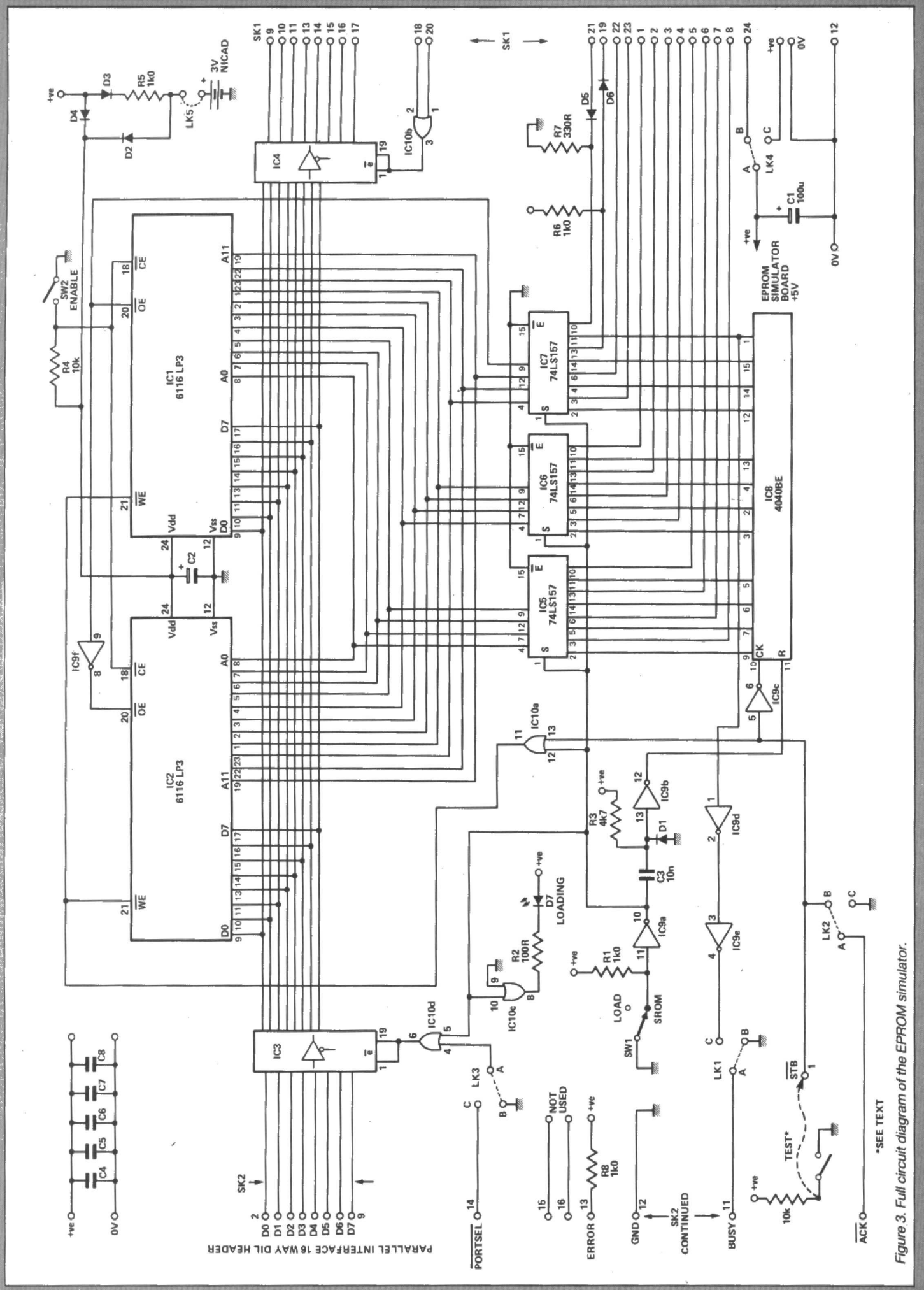
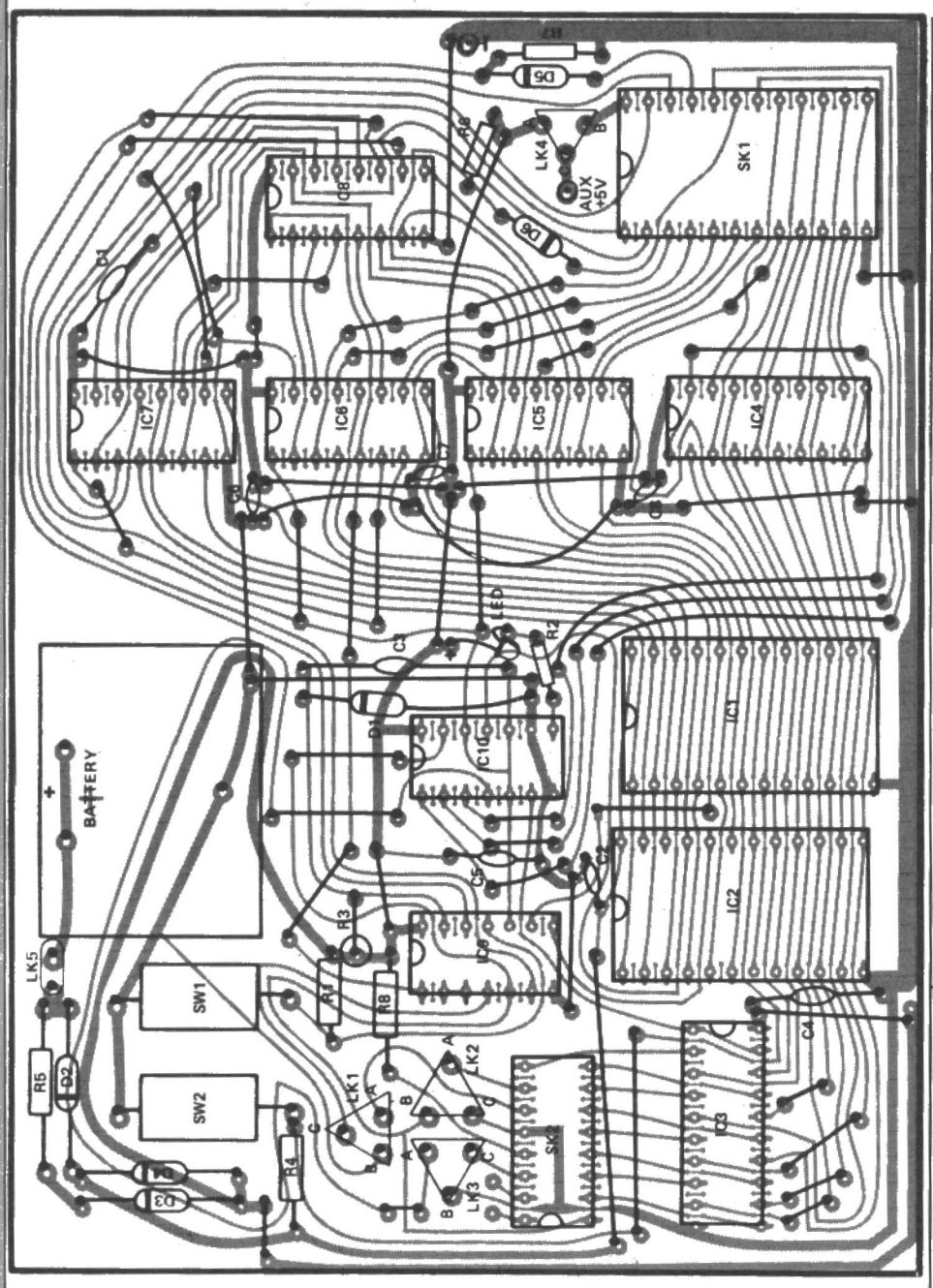


Figure 3. Full circuit diagram of the EPROM simulator.



Overlay of the EPROM simulator project. PCBs are available from E&CM – those people wishing to make their own boards should send a large SAE to our offices for the foil pattern.

static electricity before they are removed from their conducting-foam packing. Note that the orientation of IC3 and SK2 differs from the rest of the chips. The battery, switches and LED may be mounted on the board or else flying leads may be taken from the board to these components. The IC sockets should be soldered onto the board first – be sure that SK1 and SK2 are of the type that will accept their 16 and 24-pin DIL headers before you solder them in – followed by the wire links, of which there are many. The components should go in next, and links 1 to 4 wired up. It is vital that the wire link/component overlay is consulted as the building proceeds. LK5 powers up the CMOS sockets, so you might like to leave that until last. The computer end of the 14-way ribbon cable should be connected to whatever plug or socket the parallel interface uses. The normal wiring convention for the Centronics standard is shown in **Figure 4** and this is what you might reasonably expect at the computer end. The ribbon cable cannot follow this convention entirely and note too that there is the non-standard signal called PORTSEL which is intended to help non-Centronics users. The wiring of the link pads needs some care, LK4 gives the simple option of taking the 5V power supply from a source other than the EPROM socket. LK1, LK2 and LK3 control the BUSY, ACK and PORTSEL options respectively and in each case the normal link will be A to B.

Complete the board with the exception of the ICs themselves and double check the polarity of the electrolytic capacitors and the diodes. Apply +5v to pin 24 and 0v to pin 12 of SK1. Establish that the power supply is available at all IC sockets, and that on IC1 and IC2 SKTs the Vdd is about 4.3v (0.7v is dropped across D4). Remembering to turn off the power before inserting or removing any ICs, first put in IC9 and IC10 and establish that the LOAD LED comes on when SW1 is operated. The remaining ICs except IC1 and IC2 may be inserted, and a check made on the logic operation of the counter and the multiplexers by manually clocking STB using the small test circuit (shown on the main circuit diagram) and checking a few outputs of IC5 with a logic tester or voltmeter. Finally, if all seems well put in the memory ICs (just IC1 will do to start with) and begin sending test bytes from the computer to the card. Obviously if the same computer has a spare EPROM socket and can read back the data from the 6116, then the job will be made so much easier. Don't forget to test the battery back-up upon data that doesn't really matter. A newly purchased NiCad will take many hours on trickle charge to come up to full voltage.

Centronics signals

Pin 1	Stobe
Pins 2-9	Data 0-7
Pin 10	Acknowledge
Pin 11	Busy
Pin 32	Error
Pin 16	GND
Pin 33	GND
Pins 19-30	GND

Figure 4. Connections to a Centronics port.

PARTS LIST

Semiconductors

IC1, IC2	6116
IC3, IC4	74LS241
IC5, IC6, IC7	74LS157
IC8	4040 BE
IC9	74LS04
IC10	74LS32
D1-6	1N4148
D7	Min red LED

Resistors

R1	1K
R2	100R
R3	4K7
R4	10K
R5	1K
R6	1K
R7	330R
R8	1K

Capacitors

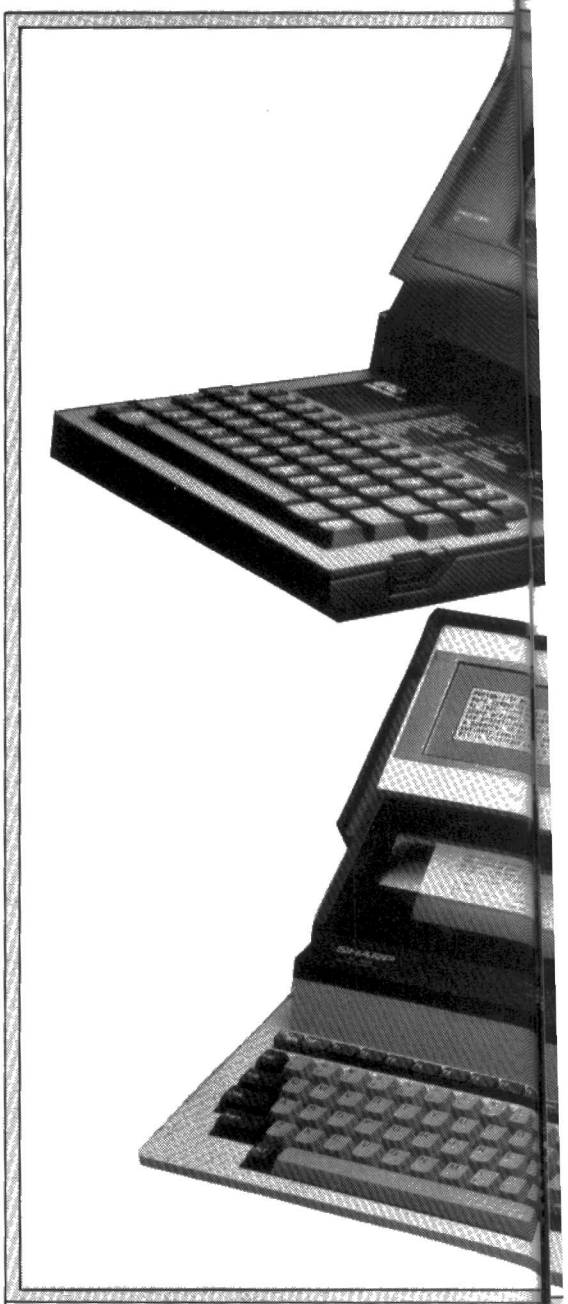
C1	100uF 16v Radial Electrolytic
C2	10uF 16v Tantalum
C3	10nF Disc ceramic
C4-10	100nF Disc ceramic

Miscellaneous

PCB; Twelve IC sockets: 3 x 24-pin, 2 x 14-pin, 2 x 20-pin, 5 x 16-pin; 3.6v NiCad battery (Ambit); Two sub-min SPST switches; Ribbon jumper-cables: 16-way single ended DIL header, 24-way double ended DIL headers (Watford); Veropins for LK1-3.

All about bubbles

Bubble memory is an exciting new technology which will soon be cheap enough for use in home computers. William Owen explains how it works.



Bubble memory is the most compact form of non-volatile memory available; it has a low power consumption and very fast access time, a large memory capacity and, unlike disc and tape, doesn't suffer from head crash. Bubble is also extremely rugged, operating efficiently in the harshest of environments, and in cartridge form – the most common available – it is exchangeable and portable. So why isn't everybody using it?

Bubble memory is the great white hope (and the great misnomer) of the computer world. For at least five years the pundits have predicted that bubble will be the next big thing: each year the bubble bursts. Despite its advantages bubble memory has proved too expensive for all but the most specialised – usually military – applications. Companies of the stature of IBM, Texas Instruments, Plessey, NatSemi and NEC have despaired of making the necessary technological breakthrough and halted research projects.

However, there are now indications that bubble memory is about to find an established place for itself, particularly in portable and small micros. For the moment bubble is still too expensive for all home and most personal computers, but it is possible, if not probable, that in future the disk drive will be replaced by the bubble cartridge interface.

But bubble memory has very little to do with bubbles: the 'bubbles' are tiny circular magnetic domains within an epitaxial film layer on the chip. The domains are made to move through the medium by an external magnetic field – they are polarised in the reverse direction to the medium.

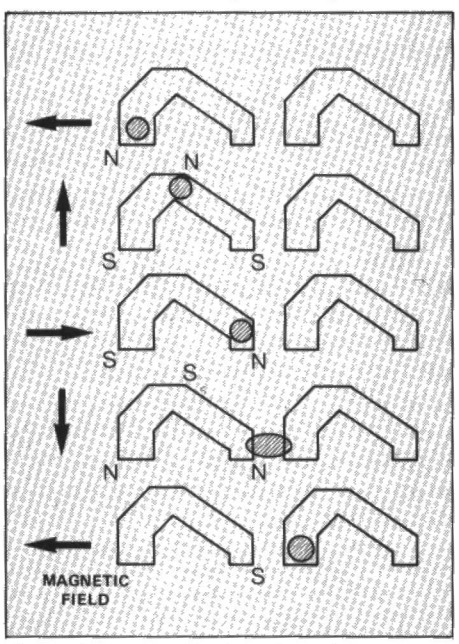


Figure 1. The way in which the bubble memory moves through the chevrons.

Modern magnetic bubble memories (MBMs) are constructed from a thin layer of garnet, deposited as epitaxial film on a substrate of simpler garnet. It is within the complex layer of garnet that the bubbles reside. Above is a layer of insulation, a conductor, a permalloy and a passivation layer.

The complex garnet has a pattern of metal shapes deposited onto its surface. The shapes are usually in the form of chevrons, and the bubbles, or magnetic domains, move through each chevron and can jump between one chevron and another under the influence of the magnetic field (see **Figure 1**). Data is represented by either the presence or absence of a bubble within a chevron: those with a bubble are ones, those with-

TABLE 1 Comparison of characteristics of disk, tape and bubble (source, Immediate Business Systems).

STORAGE DEVICE	MEAN ACCESS TIME (ms)	ERROR RATE	MTBF (YEARS)	LIFE OF ELEMENT	OPERATING TEMP RANGE °C
HARD DISK	30 – 75	10^{-10}	2	5 YEARS	+5 – +45
FLOPPY DISK	55 – 173	10_{-9}	1	0-500h	+5 – +45
TAPE CASSETTE	START/STOP TIME 20ms plus SEEK TIME	10^{-6}	1	50-500h	+5 – +45
BUBBLE	12·5	10^{-16}	>40	>40 YEARS	—30 – +70
BUBBLE CASSETTE	12·5	10^{-16}	40	LIMITED ONLY 20,000 INSERTION REMOVAL CYCLES	0 – +50

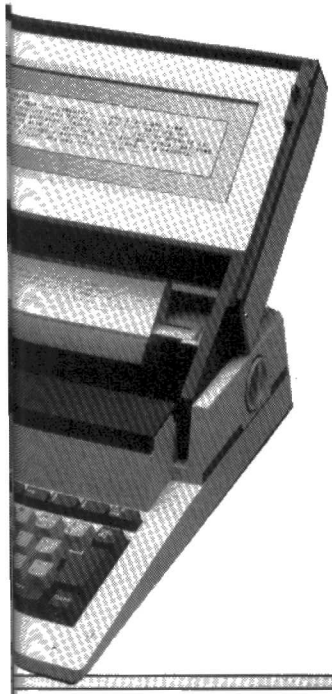
Sharp PC-5000

The Sharp PC-5000 is a 16-bit 8088 portable computer which uses 128Kbyte bubble memory cartridges as the standard means of data storage. The bubble interface acts in the same way as a single sided, single density disc drive operating under MS.DOS, allowing file transfer and information exchange betweenMBM cartridges and floppy drives.

Despite its expense (£150 per cartridge) bubble memory was chosen for the PC-5000 not only because of its efficient capacity/size ratio, but also because of its extremely low power consumption; this allows the PC-5000 to become a truly portable computer in that it is in no way reliant on mains power supplies.

The bubble memory adapter is interfaced with the host system via 8 bits of parallel data, and performs various types of controls and transfers of data through the register in the bubble memory controller.

The bubble device is a Hitachi version, which employs a major line and minor loop system, comprising two 512 Kbit blocks. Each block has 228 minor loops, which are data storage areas. Each loop has a storage capacity of 2335 bits. Of the 288 minor loops in each block, 256 are usable for actual data storage, one loop for parity data, and one loop as a marker for storing reference markers for page addresses. The remaining 30 loops are redundant, and used to replace any defective minor loop; this is common practice in bubble memory manufacture, and cuts the component failure rate by a considerable amount.



The GRiD Compass

While the Sharp PC-5000 may be just beyond the price bracket of the average home computer enthusiast (£1195) the GRiD Compass is way over the horizon: prices start at £5195.

The GRiD is very much the computer of the future. It has 384 Kbytes of bubble memory and 256 Kbytes RAM. There are two central processors, the true 16-bit Intel 8086, and the 8087 high speed maths processor. The machine includes an inbuilt modem, IEEE 488 and RS232-C interfaces, battery powered real-time clock, and flip-up electroluminescent flat panel display – all in the space of a very small brief case. The single omission is an on-board thermal printer (an omission not made by Sharp). The GRiD is marketed as 'the ultimate professional computer'. Given two years the ultimate home computer may look something like this.

out are zeros.

This goes some way to explaining how MBM is programmed, but not why it is non-volatile. The bubbles are moved through the chevrons by a rotating magnetic phasor, revolving 100,000 times a second, but the bubbles are kept in position when the power is switched off by a second, permanent magnet biased at an angle to the chip. The bubble chip itself is heavily protected from external magnetic fields which could affect the data within, while the permanent magnet holds each bubble in the place to which it was sent by the rotating phasor, whether the power is on or off.

MBM uses serial architecture, based on a number of loops of chevrons. Each loop has an input for writing (called a swap gate) and an output for reading (a replicate gate). Bubbles are generated according to data sent from the CPU, and enter the loops via THE SWAP GATE. The bubbles are pulled out of the loop at the replicate gate (for reading) by a small current pulse, and read by a bubble detector.

A typical bubble device can store one megabit of data within its loops. Access time is very fast because the loops are held in parallel series, distributing the data between them; that is, a serial stream is sent to the bubble generator and then the stream of data is deivided up between a number of loops. Typically the data rate is 100Kbits per second, and the access time 12.5 milliseconds.

Magnetic bubble emory compares favourably in terms of speed with magnetic

TABLE 2 Comparison of solid state memories (source, Immediate Business Systems).

MEMORY	NON VOLATILE	FREQUENT WRITES	ERASABLE	ERASABLE IN SYSTEM	EFFICIENT BIT TRANSFER	EFFICIENT BYTE TRANSFER
SRAM		●	●	●	●	
DRAM		●	●	●	●	
ROM●					●	
PROM	●				●	
EPROM	●		●		●	
EE PROM	●		●	●	●	
NV RAM	●	●	●	●	●	
MBM	●	●	●	●		●

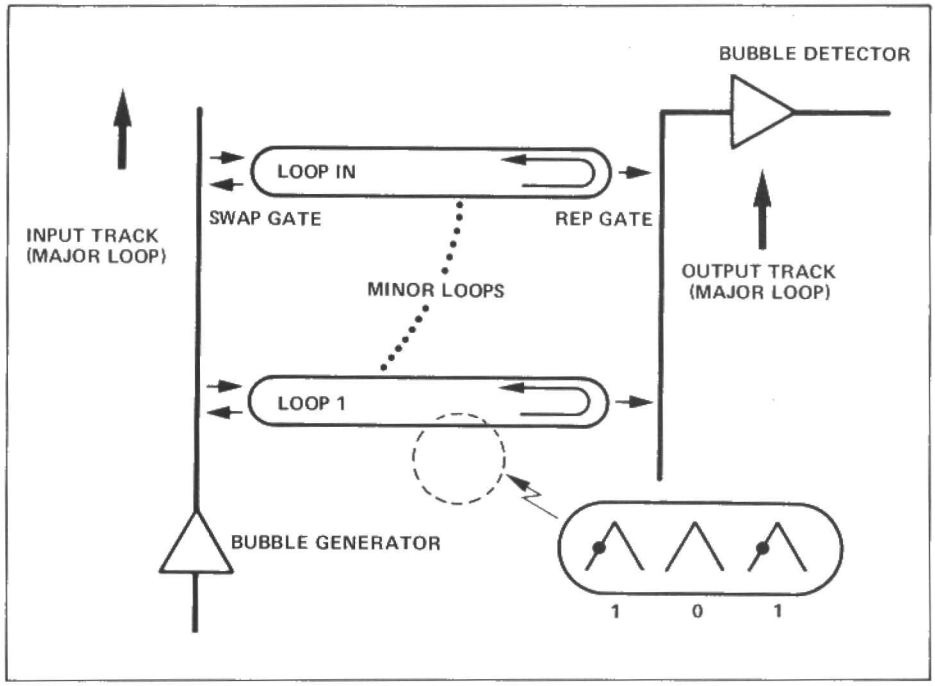


Figure 2. Simple bubble memory architecture.

disk and tape (see **Table 1**), with a performance and cost per bit intermediate between semiconductor memory and disk. Only dynamic RAM has a capacity per device equivalent to MBM.

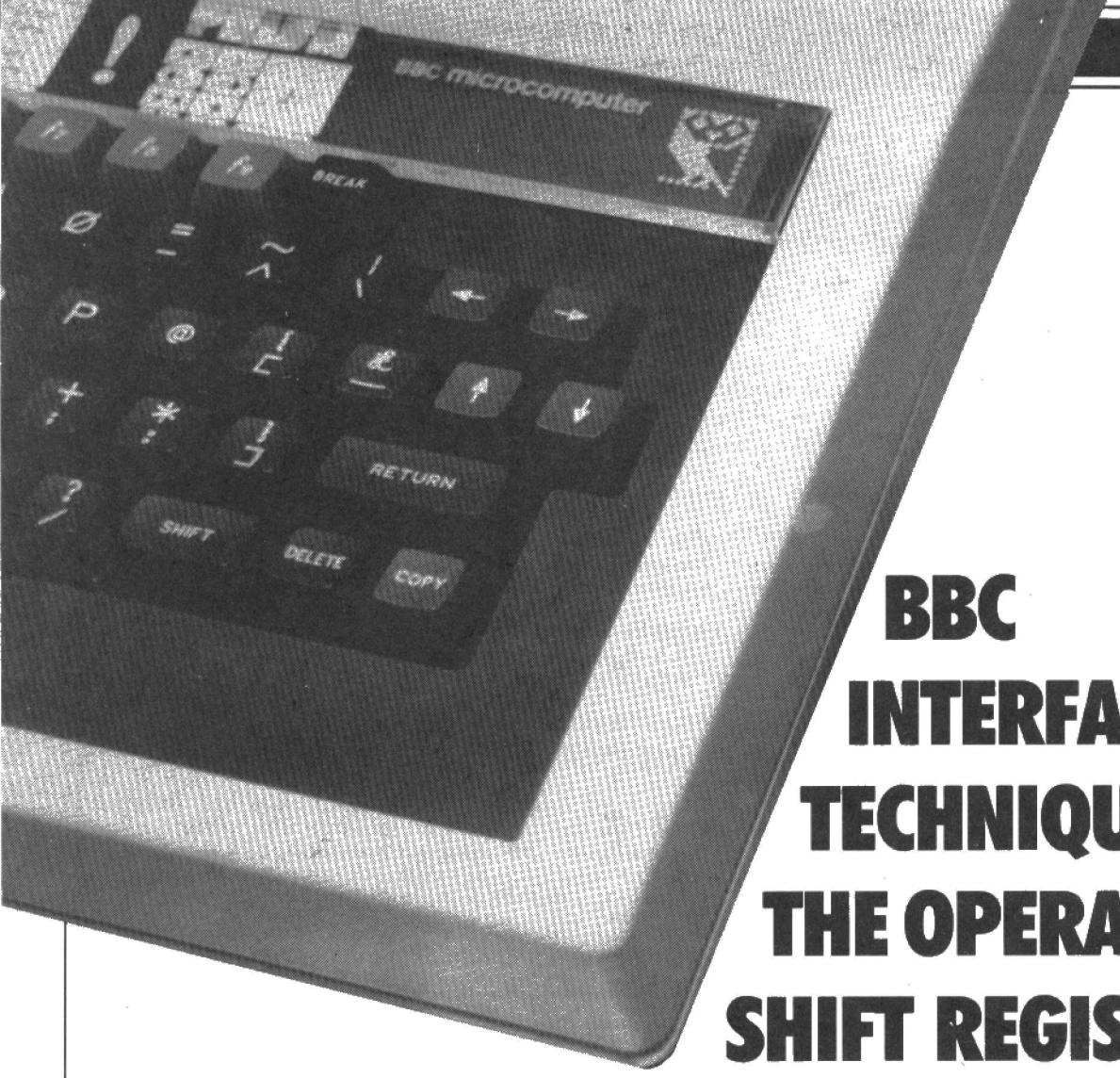
Table 2 compares the characteristics of bubble memory and other solid state devices, and it can be seen that only battery backed (non-volatile) RAM comes anywhere close to the flexibility of MBM.

The development of MBM has, to a degree, parallelled semiconductor technology in that it was dependent upon the associated recution of film thickness and improvement in integration techniques to make it a viable proposition. Bubble memory was invented in 1967 by Bobeck, of Bell Telephone Laboratories in the USA, but similar technologies involving thin film plated wire were being investigated in the fifties at Bell and Hughes Research. Full commercial production did not begin until 1979 (at Fujitsu), but already 256K devices were being produced (by Fujitsu, then Texas Instruments and Rockwell). Today the average size of MBM devices is 1MBit, and large quantities are produced by Bell, Intel, Motorola, Hitachi, Fujitsu and Sagem. Hitachi and Fujitsu alone manufacture over 25,000 1Mbit devices per month.

A number of factors have contributed to the upsurge in MBM production. Not least the reduction in manufacturing costs to a level acceptable to certain key growth areas: communications and portable computers.

The communications industry requires high speed, portability and reliability and the advantages of bubble in portable computers are obvious: compact size and ease of use.

Other applications have been found in avionics, control equipment, data loggers, office automation, portable instrumentation and shipborne instruments. But portable PCs are by far the most interesting area, and you will find details of two very recent products which point the way to the future on the previous page; the Sharp PC5000 and Grid Compass.



BBC INTERFACING TECHNIQUES – THE OPERATION OF SHIFT REGISTERS

This month, Paul Beverley describes the basic operation of shift registers and illustrates their use in a variety of applications.

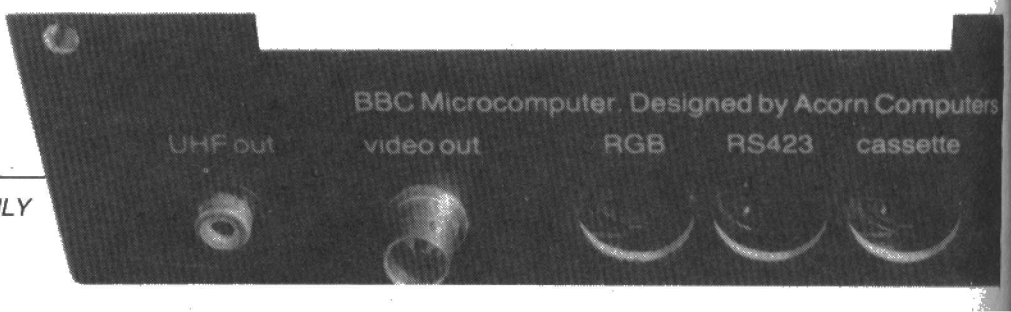
A shift register is an electronic circuit capable of storing a binary number, but which also is capable of being shifted ie the number can be moved sideways, being pushed along one bit each time it receives a clock pulse. By shifting sideways, the data can be entered or sent out serially.

When shift registeres are supplied as individual chips they come in a whole variety of configurations depending on the different ways in which they can be written to and read from. They must at least have some sort of serial input or output, or there would not be much point in calling them shift registers, yet some of them such as the 74LS91 only have serial input and output with no parallel access at all. Others allow you to write into them in parallel on eight data lines and then shift out serially (PISO – parallel in, serial out), and others allow numbers to be shifted in and then read out in parallel using the eight data lines. (SIPO – serial in, parallel out).

The main limitation on these single chip shift registers is the number of pins on the chip. If you want parallel input as well as parallel output then you have to have 16

Figure 1. The modes of operation of the VIA shift register as controlled by bits 2, 3 and 4 of the auxilliary control register (&FE6B on the BBC microcomputer).

BIT								
7	6	5	4	3	2	1	0	Operation
X	X	X	0	0	0	X	X	Shift register disabled
X	X	X	0	0	1	X	X	Shift IN under control of timer 2
X	X	X	0	1	0	X	X	Shift IN under control of system clock
X	X	X	0	1	1	X	X	Shift IN under control of external clock
X	X	X	1	0	0	X	X	Shift OUT under control of timer 2 (free-running)
X	X	X	1	0	1	X	X	Shift OUT under control of timer 2
X	X	X	1	1	0	X	X	Shift OUT under control of system clock
X	X	X	1	1	1	X	X	Shift OUT under control of external clock



LISTING 1 – Outputting characters, typed in at the keyboard, from the VIA shift register.

```
10  ?&FE6B = &14
20  ?&FE68 = 10
30  REPEAT
40    N% = GET
50    IF N% = 13 PRINT ELSE PRINT
      CHR$(N%);
60    ?&FE6A = N%
70    REPEAT UNTIL (?&FE6D AND 4)
      = 4
80    UNTIL0
90  END
```

lines for the parallel data even before thinking about serial lines, control lines or power supply. The 6522, however, already has parallel input and output on the normal 6502 data bus. Thus all it needs is a serial in/out line and a control line. This facility amongst others is provided by the CB1 and CB2 control lines. The VIA's shift register can then have numbers fed in and taken out in parallel as well as in serial form. (PIPOSISO!) The parallel data is accessed through register 10 of the VIA (&FE6A on the BBC microcomputer) and control of the modes of operation of the shift register is achieved through the auxilliary control register (register 11 – &FE6B on the BBC microcomputer).

Whether the data is being shifted in or out, CB2 is used to carry the data. This means that the same physical line may be used for output as well as for input, which is exactly the same as the way that each individual port line can be used for either input or output. Care has to be taken, therefore, in programming the shift register because if you are trying to shift data in, and have CB2 connected to a source of signals, you might accidentally set CB2 into an output mode. This might well damage one of the ICs either the 6522 itself or the chip which is driving signals into it.

The same is true of the CB1 line which is used to carry the clock signals which control the shifting of data into or out of the shift register. The problem here is probably more serious than with CB2 as it is easier to confuse the different modes and end up with CB1 acting as an output when you intended to feed clock pulses into the VIA.

Mode selection

The mode of operation of the shift register is controlled by bits 4, 3 and 2 of the auxiliary control register (register 11, &FE6B on the BBC microcomputer). Bits 7, 6 and 5 of this register, which we dealt with in a previous article in this series, control the modes of operation of the two timers. Bits 1 and 0, which control the latching of data into the two ports, will be referred to in a later article. **Figure 1** shows in tabular form how the different modes are controlled, and as you can see, bit 4 is what might be referred to as the data direction bit. As with the individual bits in the data direction registers for the two ports, logic 1 sets CB2 as an output, and logic 0 sets it as an input.

There are three basic ways of clocking the data, as set by bits 3 and 2, for each of the states of bit 4. "01" selects timer 2 as the source of clock pulses, "10" selects the system clock, and "11" allows you to apply an external clock pulse to CB1.

In the 01 mode, the low byte of timer 2 is used by the VIA to time the clocking of the shift register. If the number in the timer 2 low order latch is N, then the register is shifted once every 2 * (N + 2) clock cycles. This allows a time of between 4 microseconds per bit and 514 microseconds per bit, assuming a 1MHz clock rate since N can have values between 0 and 255. To set up this mode, all you have to do is to set the relevant bits of the ACR and put the delay time value into the timer 2 low latch (&FE68 on the BBC microcomputer).

"Data can be loaded into the VIA shift register on one machine and shifted along a single communicator channel to a second VIA which shifts the data in".

In the 10 mode the register is shifted once every other cycle of the system clock ie once every two microseconds, and in both of these modes, CB1 acts as an output, producing a pulse which can be used to clock an external shift register whose data line is attached to CB2, CB1 being inverted every system clock pulse. The actual shifting takes place on the positive-going edge of the CB1 clock pulse.

The 11 mode allows a pulse to be fed from an external source into CB1 to control the shifting at any speed you wish. You could, for example, generate a suitable pulse train by setting timer 1 into the free-running mode and taking an output from PB7. As with the other two modes, clocking takes place on the positive-going edge of the clock pulse applied to CB1.

Of the two modes not so far mentioned, 000 is the simpler. It totally disables the shift register and allows the CB1 and CB2 lines to be under the control of the peripheral control register for use in the handshaking of parallel data transfer. The 100 mode on the other hand is an active mode. Since bit 4 is set, it is an output mode, and what actually happens is that any number loaded into the shift register is outputted serially on CB2 under the control of timer 2 as in the 101 mode, but it is outputted repeatedly. What happens inside the VIA is that the serial output of the shift register is fed into its serial input so that the number just cycles round and round and the data comes out repeatedly. I cannot think of any applications for this myself. Anyone got any ideas?

However, to try this mode out, a buffered loudspeaker could be attached on the CB2 line and the following program executed:

```
?&FE6B=&10
REPEAT ?&FE6A = RND
?&FE68 = RND
UNTIL 0
```

This will produce digital noise by changing the number being outputted and also the rate at which it is being clocked.

Flags and Interrupts

Bit 2 of the interrupt flag register (register 13 – &FE6D) acts as a flag, to signal the completion of 8 shifts, either in or out. Your program can therefore wait until the shifting is complete before reading the shift register. This flag is set whenever eight shifts have taken place whether the clock is internally or externally generated.

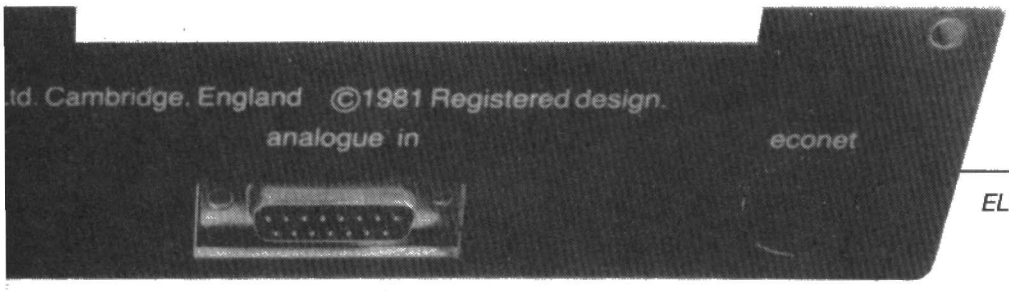
Alternatively, this flag can be used to generate an interrupt, providing it has been enabled by using the interrupt enable register (register 14). To enable an interrupt on completion of 8 shifts, you simply load &84 (10000100 in binary) into the IER. (?&FE6E = &84 on the BBC microcomputer.) Bit 7 has to be set, to indicate that you want to enable the interrupt and bit 2 is set to indicate which interrupt is to be enabled. To disable it again, you put &04 into the interrupt enable register.

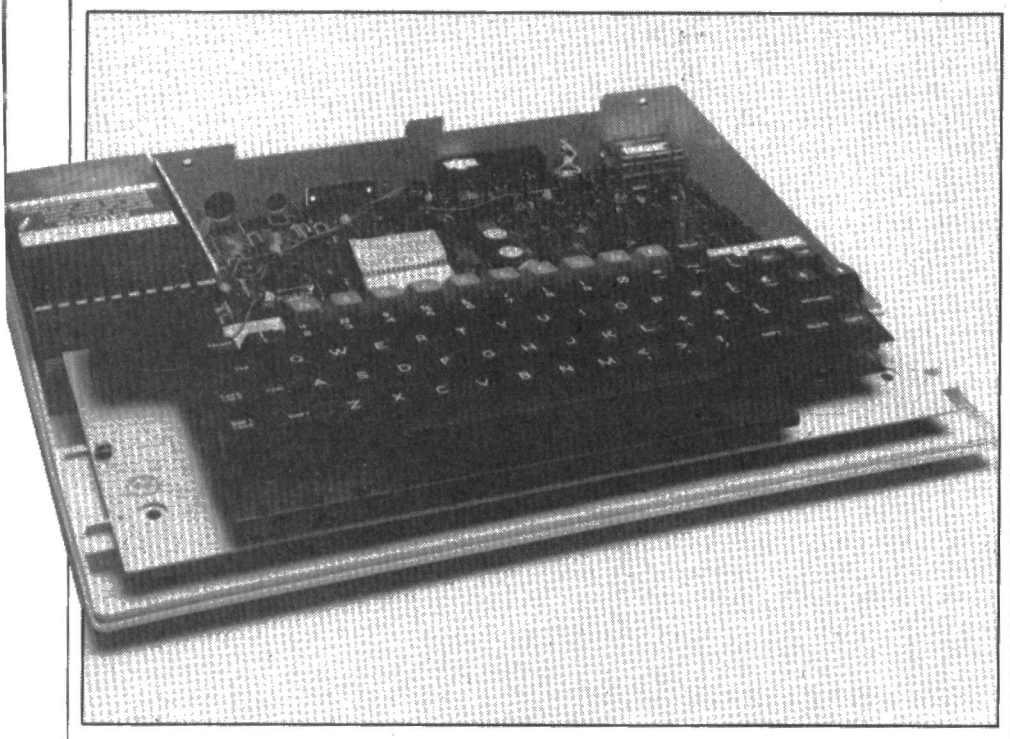
Applications

All that has been said so far could have been gleaned directly from the 6522 data sheets, even if they do make rather heavy reading. What is probably more helpful is to give a couple of applications so that you can see how the shift register can be used in practice. As in previous articles, we will be dealing specifically with the BBC microcomputer, but similar programs could be written for other 6502 based microcomputers.

Serial communication

It is possible to use the VIA shift register for serial communication of data. In other words, data can be loaded into the VIA shift register on one machine and shifted out along a single communication channel, such as a wire, to a second VIA which shifts





the data in. The clock signal from the sending microcomputer also has to be transmitted in order to synchronise the clocking of the data into the shift register at the other end.

Programs 1, 2 and 3 enable you to try this out. The CB1 lines of the VIA's on the two microcomputers have to be linked together, as do the CB2 lines, and there must of course be an earth return ie you must link together the 0 volt lines of the User Port connectors of the two microcomputers. The sending program, (program 1) is loaded into one computer and the receiving program (either program 2 or program 3) into the other. The sending program sets the shift register into mode 101 (at line 10, &14 = 00010100 in binary) to put it into the send mode, controlled by timer 2. The time delay value is set in time 2 low latch (line 20) and it then picks up characters from the keyboard (line 40) and outputs them by putting them into the shift register (register 10, &FE6A on the BBC microcomputer, line 60) which initiates the shifting process. The program then waits until the VIA interrupt flag is set, indicating that the shifting process is complete (line 70) before getting the next character.

At the receiving end, the VIA shift register is put into the 011 mode ie shifting is under control of an external clock. This clock pulse, of course, comes from the CB1 line of the sending microcomputer. **Listing 2,** written entirely in BASIC, waits until the shift register flag is set, to indicate the completion of an 8 bit shift (line 40) before reading the shift register (line 50). If the code received is not a control code, then it is printed, but if it is a carriage return, a new line is generated. The reason for doing this is that if an erroneous code is received which happens to be a control code, it could produce all sorts of undesirable effects. Line 20 also needs explanation – unless a read is done, the shifting process does not start, therefore we do a dummy read in order to get things going.

Listing 3 is a machine code version of listing 2 which was tried because certain problems occurred with **Listing 2.** Characters were being incorrectly received, and this appeared to be caused by the loss of synchronisation of the signal. Using machine code will obviously increase the speed of response to in-coming signals, but it did not in fact reduce the frequency of errors. Even at the slowest speed, as set by timer 2, there were still errors. The most likely cause of this problem is the degradation of the waveform caused by the capacitance of the wires carrying the signals. According to the 6522 data sheets the timing of the clocking of the shift register should be such that the delay between the positive-going edge of the clock pulse and the next negative-going edge of the system clock should be between 0.3 and 1.0 microseconds. Thus the capacitive loading on the line may well increase the delay sufficiently to cause the clocking to be a bit marginal.

In the experimental set-up there were ribbon cables at both ends going into ready-made interface boxes with 4mm sockets. The CB1 and CB2 lines were then linked by 4mm plug leads. When these were replaced by two ribbon cable jumper leads wired together directly, the error rate decreased, but communication was still not error-free. This supports the view that the errors are a result of the degradation of the waveform.

The solution would be to use a line driver IC as close to the sending VIA as possible. However, this was not actually tried out as, on the BBC microcomputer at least, this application is little more than an academic exercise. The BBC microcomputer has RS423 serial communication hardware with excellent, interrupt driven software in its operating system which enables it to operate in send and receive modes simultaneously.

LISTING 2 – Receiving and displaying characters coming in serially through the VIA shift register, but excluding control codes.

```
10  ?&FE6B = &C
20  DUMMY = &FE6A
30  REPEAT
40    REPEAT UNTIL (?&FE6D AND 4)
50    X% = ?&FE6A
60    IF X% > 31 PRINT CHR$(X%);
70    IF X% = 13 PRINT
80    UNTIL0
90  END
```

LISTING 3 – Machine code version of program 2 except that all codes that are received are sent to the operating system VDU routines whether they are control codes or not.

```
10  ?&FE6B = &C
20  P% = &C00
30  [
40  .START
50  LDA &FE6A       Dummy read
60  LDA #4
70
80  .WAIT           Wait for
90  BIT &FE6D       shift register flag
100 BEQ WAIT        to be set.
110
120 LDA &FE6A       Get character.
130 JSR &FFEE       Output to screen.
140
150 JMP START       Do it again.
160 ]
170
180 CALL START
```

Hardware Random Number Generator

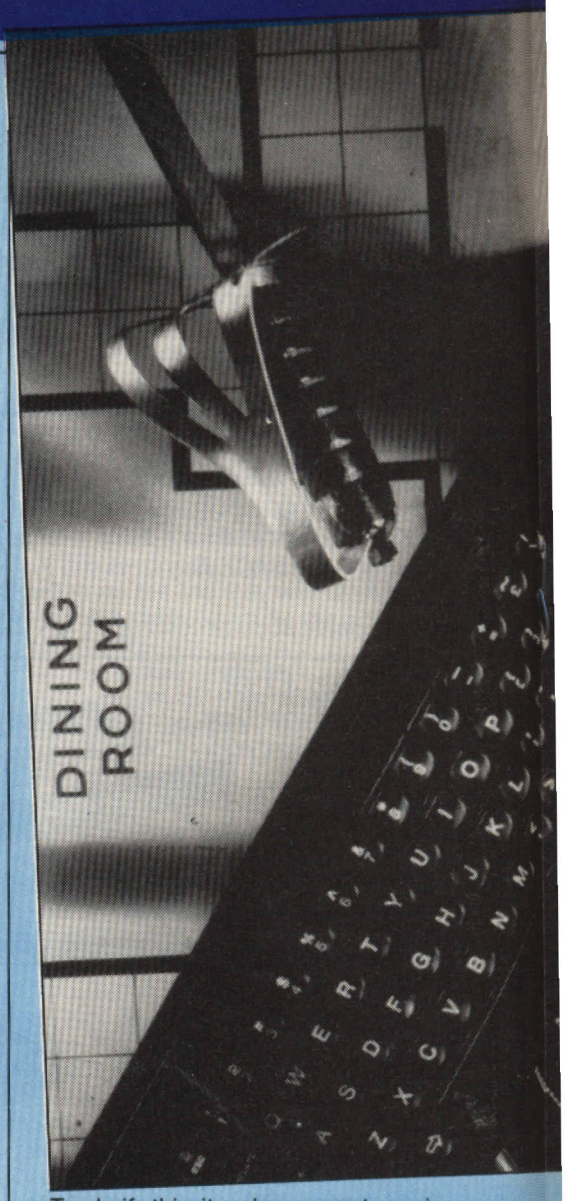
One application to mention here is that of using the VIA to read in the data from a hardware random number generator. The random number generator in BBC BASIC is implemented in software using a machine code routine which is the analogue of an electronic circuit known as a ring counter which is basically a shift register (33 bits long!) with some exclusive-OR feedback. If this ring counter were implemented in hardware it could be read, as in the previous application, by using the VIA shift register.

Next month

A much more useful project for the BBC microcomputer owner who is interested in electronics is that of making a logic state monitor. This is a rather grand name for a simple device that reads the logic states of the number of lines and in some way displays them. Next month we will look at a way of doing this with the BBC microcomputer.

Extending QL SuperBasic

Adam Denning, of Micronet 800, shows that QL Basic can be enhanced by way of extensions to the machines procedure list.



Although the QL's SuperBasic is in many respects a wonderful programming language it is somewhat deficient when seen as the QDOS command language. Where are essential disc commands like BACKUP and RENAME?

Luckily the mechanism for extending the procedure list is simple – in Basic! The procedures to do these extra operations are also generally easier to write in Basic, so that's what we're going to do here.

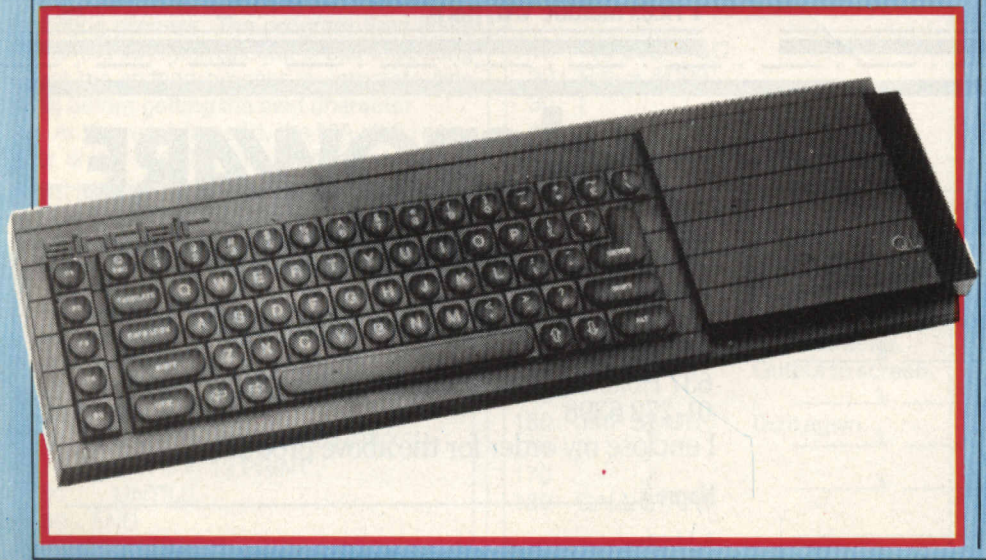
The author is currently working on the machine code versions of these procedures so that these can be linked in as part of the 'permanent' procedure/function list on boot. The QDOS documentation explains reasonably clearly how to do this, but the lack of a QL assembler makes things rather awkward at the moment.

When adding procedures and functions to SuperBasic from **within** SuperBasic, one has to choose a line numbering such that as little conflict as possible occurs with the user's own program and the easiest way to do this is by using the highest line numbers possible. The list of procedures described here starts at line 31500, and with the default AUTO line number of 100 it would take a very extensive program indeed to overrun the procedures. Naturally those procedures that are not required within a particular application may be deleted if the program does get too big, but if you do this you must ensure that procedures that are called by other procedures are not deleted.

As a number of QL owners are likely to be new to Basic programming it is as well to get the distinction between procedures and functions clear at the start. In many respects they are the same in that they are both defined blocks of code that perform a particular operation or sequence of operations.

"... the mechanism for extension is simple".

The fundamental difference is that a function returns a result while a procedure does not. This issue can become a little clouded when a procedure is written which alters the value of a variable *declared outside the procedure,* as it then appears to return a result, but in such cases the result is implicit rather than explicit.



To clarify this, it makes sense to assign a variable to a function, but a similar operation within a procedure would be syntactic nonsense. In other words if **func(param)** is a function then one can write

```
result = func(param)
```

but if it were a procedure one could not. As an example, let's consider some SuperBasic functions and procedures. **SIN, INT, RESPR** and **CHR$** are examples of functions as all the lines below make perfect sense:

```
PRINT SIN(PI)
x=INT(RAD)(45))
space=RESPR(1024)
a$=FILL$(CHR$(26),34)
```

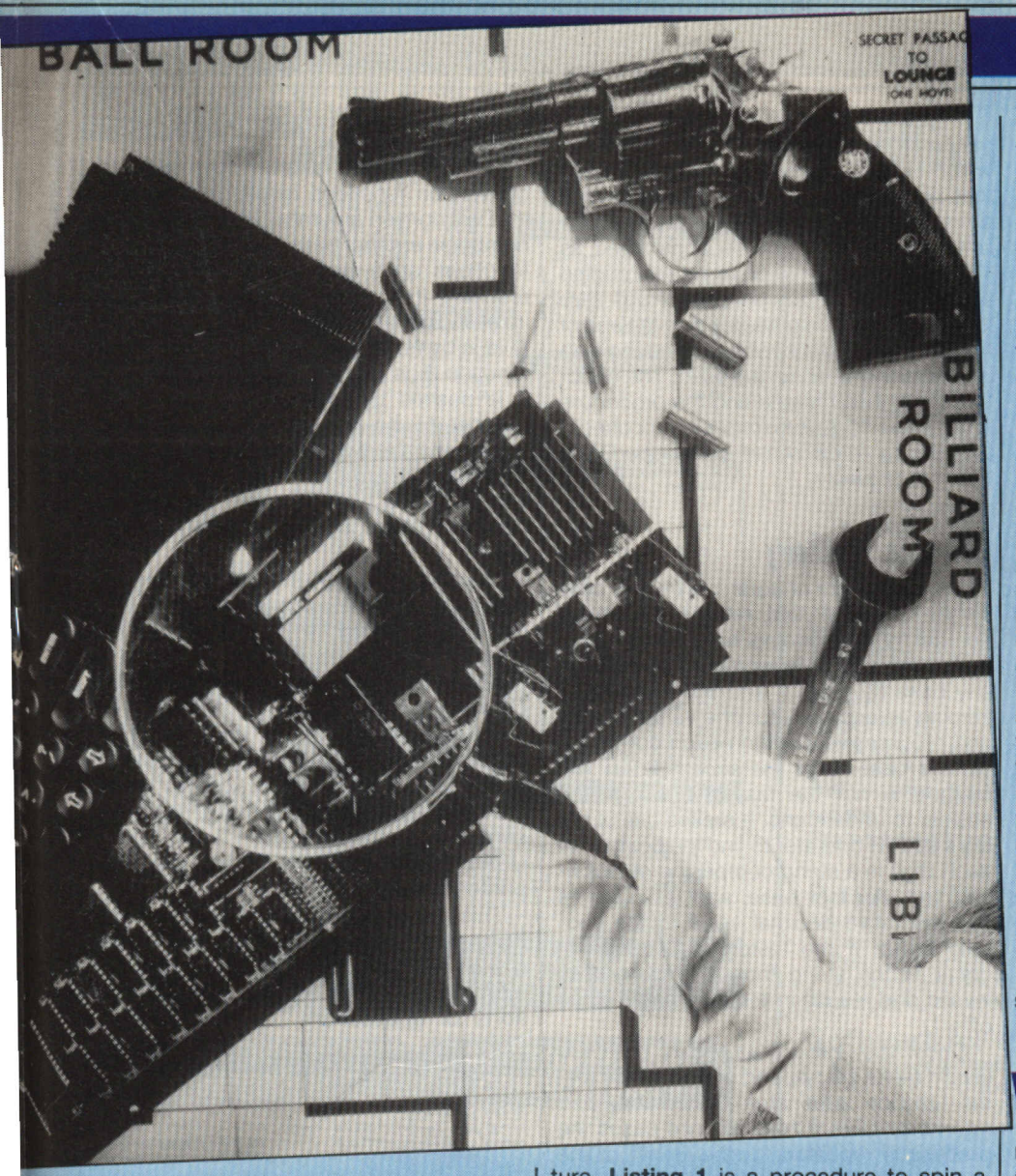
SuperBasic procedures are things like **CLS, PRINT, LIST** and **POKE.** That's because you can say things like:

```
PRINT "I'm a procedure"
```

while things like

```
a=CLS
```

are pure rubbish! Notice how SuperBasic functions, whether inbuilt or user defined, *always* have their parameters in brackets while procedures in general don't. We say 'in general' because it is quite valid to include parameters in brackets, but doing so alters the way they are treated by



SuperBasic. A parameter of x is subtly different to a parameter of (x) – the first case results in the variable itself being the parameter while parenthesised parameters pass the value.

This is only of significance to us when a procedure alters a global variable – by passing the variable in brackets it won't get altered. Sometimes!

While we're defining things we might as well make the distinction between operators and procedures/functions too. An operator returns a result by operating on one or more operands, so that in an operator that returns the negative of the operand following it, and + returns the result of adding the operands on either side of it and **MOD** produces the remainder of an integer division of the left hand operand by the righthand operand. Notice that, unlike most other computers that supply this function, **INSTR** in SuperBasic is an operator and not a function.

Now onto our defined procedures and functions. Notice that in many cases we have declared certain of the parameters as being typed (ie said that they are integer, string or whatever) despite the fact that formal parameters are defined as typeless in SuperBasic. This has been done simply to clarify the use of each parameter and it is not necessary to adhere to this nomenclature. **Listing 1** is a procedure to spin a specified microdrive cartridge until the ESCAPE key is pressed. Although this may at first sound entirely fatuous it has proved to be of immense value when trying to recover data from microdrive cartridges that seem intent on always returning **bad or changed medium** error reports. It will by no means always recover a set of data but a good spin of sixty seconds or so does seem to do a cartridge wonders! The procedure also raises a few interesting points about SuperBasic.

As with other procedures it is good programming practice to reduce the scope of variables within a procedure as much as possible – in other words, make them local as the first line of a procedure or function definition. This has been extended to include loop identifiers (used in REPeat, FOR and SELect) because the scope of these matches are that of a variable.

The **SPIN** procedure first builds up a string to represent the specified microdrive cartridge, putting the result in **drive$.** It then opens the smallest screen window possible to channel 200. Such a high channel number was chosen simply because it is unlikely to conflict with channel definitions elsewhere in a program. This line also shows up to points worth noting. Firstly, although the QL User guide implies that window definitions can use steps of 1 pixel this in fact seems to be 2 pixels in reality. Secondly the guide again implies that only channel numbers #0 to #15 are valid in Basic (as is the case with the Spectrum). This also turns out to be untrue, although we have yet to discover the highest channel number that is available.

Once this window is open we enter a loop in which the cartridge is repeatedly DIRd to this channel until the ESCAPE key is pressed. As the window is so small and out of the way it is, to all intents and purposes, invisible. Once ESCAPE has been pressed we clear and close channel 200 and then return to whatever called the procedure. Believe me, it really isn't as facile as it seems!

The procedure of **Listing 2** is short and sweet and merely serves to increase the friendliness of SuperBasic. It allows us to type **CAT n** to catalogue a drive rather than having to go through the eventually rather tedious process of having to type **DIR mdvn_** each time. As the procedure stands it can only catalogue to the default channel (1) but it is a simple matter to alter it to accept a second parameter representing the channel to catalogue to.

The procedure is straightforward and self explanatory.

What's in a name

Listing 3 shows a procedure that is of rather more immediate use and simply adds a **RENAME** procedure to SuperBasic. It has three parameters – the file name to be renamed, the new file name and the drive. As it stands the procedure will not allow you to generate a copy of a file on a different drive, which is standard RENAME practice. If you really want to do this, try the QL's COPY procedure! RENAME works by building up two strings – one to define the old file and one to define the new file. The old file is then copied to the new file and subsequently deleted, which is the nearest we can get to a RENAME. Certainly its effect is to rename a file, but if the file to be renamed is longer than the space left on a cartridge it obviously can't do its job. It would probably be possible to write a machine code procedure to rename every sector of a file, but it would be of dubious value as it would decrease the life of the cartridge, which is short enough now!

Note that there is no explicit error checking within the procedure, so if the old file does not exist or the new file does already exist, a standard SuperBasic error message will be generated and the procedure will be aborted. For this reason error

"SuperBasic is somewhat deficient when seen as the QDOS command language".

checking was deemed unnecessary.

The most complicated procedure is that which performs a drive-to-drive backup. It uses a fairly devious technique to discover which files exist on the source drive's cartridge, and it includes the option to delete the original files as it goes. This option would be taken if a cartridge were proving unreliable and you wanted to reformat it but keep the files it contains intact. This option is selected by making the third parameter anything other than zero. A value of zero for **switch%** forces the procedure to perform the more usual backup facility in which files are simply copied across. This procedure is shown in **Listing 4.**

The first thing this program does is define three strings – **d$, a$** and **b$.** The first holds the filename 'dir_tmp' on the destination drive, the second holds the source drive definition and the last holds the destination drive definition. Channel 3 (you may alter this of course) is then opened as a new file and the source drive is catalogued to it. This file is then closed and re-opened for read only. Two dummy strings are then input from the directory, the first representing the cartridge name (which we don't need) and the second representing the number of sectors available on the cartridge. Again, we do not need this information. Everything left in the file is then a file name to be copied across, so we enter a REPeat loop (controlled by **movefiles**) to copy each file. The first thing this loop does is check to see whether we have reached the end of the directory file, and if so leave the file. The end of the file is checked first because if we attempted to backup an empty drive this occurrence would be trapped without error before it happened. In other words the loop is set up to act as a zero-case trap loop, something which is tedious to implement on a BBC Micro!

LISTING 1

```
31500 DEFine PROCedure SPIN(dr%)
31510 LOCal loop,drive$
31520 drive$="mdv"&dr%&"_"
31530 OPEN#200,scr_2x2a0x0
31540 REPeat loop
31550   DIR#200,drive$
31560   IF INKEY$=CHR$(27):EXIT loop
31570 END REPeat loop
31580 CLS#200:CLOSE#200
31590 END DEFine
```

LISTING 2

```
31600 DEFine PROCedure CAT(dr%)
31610 LOCal f$
31620 f$="mdv"&dr%&"_"
31630 DIR f$
31640 END DEFine
```

LISTING 3

```
31650 DEFine PROCedure RENAME(old$,new$,dr%)
31660 LOCal a$,b$
31670 a$="mdv"&dr%&"_"&old$
31680 b$="mdv"&dr%&"_"&new$
31690 COPY a$ TO b$
31700 DELETE a$
31710 END DEFine
```

LISTING 4

```
31720 DEFine PROCedure BACKUP(dr1%,dr2%,switch%)
31730 LOCal a$,b$,d$,f$,movefiles
31740 d$="mdv"&dr2%&"_dir_tmp"
31750 a$="mdv"&dr1%&"_"
31760 b$="mdv"&dr2%&"_"
31770 OPEN_NEW#3,d$
31780 DIR#3,a$
31790 CLOSE#3
31800 OPEN_IN#3,d$
31810 INPUT#3;f$;f$
31820 REPeat movefiles
31821   IF EOF(#3):EXIT movefiles
31830   INPUT#3;f$
31840   PRINT#0;"Copying "!f$
31850   COPY a$&f$ TO b$&f$
31860   IF switch%:DELETE a$&f$
31880 END REPeat movefiles
31890 CLOSE#3
31900 DELETE d$
31910 END DEFine
```

BACKUP then prints a message to channel zero (the command channel) telling us which file it is copying, and then it copies that file from source drive to destination drive. Finally, if **switch%** was given a non-zero value then it will be considered as **TRUE,** so the source drive file just copied will be deleted. If the parameter was zero then no files will be erased. Once the end of the directory file has been reached the loop is ended and channel 3 closed. The directory file is then deleted and the procedure ends. Again there is no explicit error checking, as common sense and SuperBasic ought to prevail!

Classic coding

The next piece of code is a classic conversion function that returns a decimal number corresponding to a string parameter that is taken to represent a number in any base between 2 and 36. It works by zeroing an accumulator (the variable total) and progressively adding each converted digit to base times the accumulator. The method is not the fastest but it is the easiest to understand. It makes use of the QL's **INSTR** operator to find the position of each digit in a string comprising all possible digits for that base, and returning its representative decimal value to the variable **temp.** If the tested digit does not occur in the string then it is treated as invalid and results in **temp** having a value of —1.

Once the value has been obtained invalid digits are trapped in such a way that the function terminates immediately and returns a value of zero. In valid cases the process is repeated for the length of the number string. **Listing 5** shows the procedure.

The reverse of this function is provided by the next function **OFDEC$.** This is passed as a decimal number and a number between 2 and 36 to represent the base. The decimal number is then converted to a string representing that number in the given base, and this is returned as the function's result. It works by using a process analogous to splitting a number up into the number of units, tens, hundreds, thousands and so on that form the number, but instead of splitting it into powers of ten it splits it into powers of the specified base. The resulting number is then converted to ASCII and added to the front of an initially null accumulator string (unfortunately called **hex$** here!). The end effect is to produce a string that accurately represents the number. **OFDE$** is shown **Listing 6**

Numbers to the base 16 (hexadecimal or just hex) are so useful in computing that a function to allow us to use hex numbers within a SuperBasic program would be very useful. Our **DFC** function would do this quite happily but it always has to be passed two parameters the nex number and 16 to represent the base. So this short function, called HEX, calls up DEC and automatically adds the base as 16 during the call. This means that we can now use hex numbers very easily, like so:

POKE_W HEX ('20000'), HEX('4E75')

Our final manipulation of the two base conversion functions results in a routine that will convert a number in one base to a number in another base. We use one of our functions as a parameter to the other, resulting in a base to base conversion involving decimal as an intermediate stage. This is by far the easiest method, and the function thus defined is known as **BASE$.** It is passed by three parameters – the number to be converted (passed as a string) and the two bases. The first is the base that we are converting from and the second is the base to which we want to convert. This procedure is shown in **Listing 8.**

The final procedure shown in **Listing 9** is purely of interest to those with an Epson dot matrix printer. It opens a specified channel to the printer (which is assumed to be connected to serial port one) and then prompts for an option string. This option string can consist of any combination of the digits 0, 1 and 2. Option 0 initialises the printer by sending ESC @, option 1 sets the UK character set (so that £ signs get printed properly) and option 2 sets up the USA character set (so that the # sign gets printed correctly).

This set of nine procedures and functions merely serves to demonstrate the power of SuperBasic, and the user can easily write a whole bunch more for all sorts of applications.

LISTING 5

```
31920 DEFine FuNction DEC(num$,base)
31930 LOCal total,temp,loop
31940 total=0
31950 FOR loop=1 TO LEN(num$)
31960   temp=(num$(loop) INSTR "0123456789ABCD
      EFGHIJKLMNOPQRSTUVWXYZ"(1 TO base)
)-1
31970   IF temp<-1
31980     total=total*base+temp
31990   ELSE
32000     total=0
32010     EXIT loop
32020   END IF
32030 END FOR loop
32040 RETurn total
32050 END DEFine
```

LISTING 6

```
32060 DEFine FuNction OFDEC$(number,base)
32070 LOCal num,t1,t2,hex$,buildstring
32080 num=number:hex$=""
32090 REPeat buildstring
32100   IF num<base:EXIT buildstring
32110   t1=INT(num/base):t2=t1
32120   t1=num-t1*base:num=t2
32130   hex$=CHR$(t1+48+7*(t1>9))&hex$
32140 END REPeat buildstring
32150 hex$=CHR$(num+48+7*(num>9))&hex$
32160 RETurn hex$
32170 END DEFine
```

LISTING 7

```
32180 DEFine FuNction HEX(hex$)
32190 RETurn DEC(hex$,16)
32200 END DEFine
```

LISTING 8

```
32210 DEFine FuNction BASE$(num$,base1,base2)
32220 RETurn OFDEC$(DEC(num$,base1),base2)
32230 END DEFine
```

LISTING 9

```
32240 DEFine PROCedure EPSON(chan%)
32250 LOCal uk$,usa$,init$,choice$,ch,loop
32260 uk$=CHR$(27)&'R'&CHR$(3):usa$=uk$
32270 OPEN#chan%,ser1c
      (1 TO 2)&CHR$(0):init$=CHR$(27)& @'
32280 REPeat loop:INPUT#0;'Initialise
      (0), Set UK (1) or Set USA (2)?'!choice$:I
F choice$<>'':EXIf loop
32290 FOR loop=1 TO LEN(choice$)
32300   ch=choice$(loop)
32310   SELect ON ch
32320     =0
32330       PRINT#chan%;init$;
32340     =1
32350       PRINT#chan%;uk$;
32360     =2
32370       PRINT#chan%;usa$;
32380     =REMAINDER
32390       PRINT#0;'Invalid parameter'!loop!'!'
32400   END SELect
32410 END FOR loop
32420 PRINT#0;'Printer channel'!chan%!'now set up'
32430 END DEFine
```

AMSTRAD CPC 464 FACT SHEET

In the first of a new series, Robert Penfold collects together information of a variety of computer. He begins with the new Amstrad Computer

Although the CPC464 is powered from a non-optional monitor, the use of separate monitor and power connectors enables, for example, the colour monitor to be used as the power source and a high definition monochrome monitor to be used for wordprocessing in the 80 column mode. The monitor socket is a standard 6 pin DIN RGB type (see **Figure 1**).

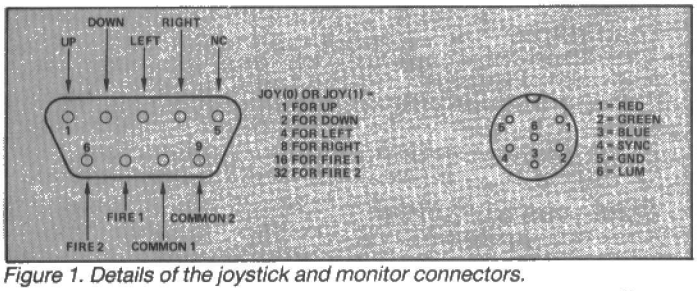


Figure 1. Details of the joystick and monitor connectors.

The joystick port is a pseudo-Atari type, and a similar joystick can be plugged direct into this 9-way D plug. It also has the ability to take a second joystick (which will plug into the first if the proper Amstrad types are used). The second joystick uses the same "direction" inputs as the first but it has a separate "common" connection. No games paddle inputs are included, but there is a provision for two fire-buttons per joystick. The joysticks can be read from BASIC using the JOY function. Details of the joystick port connections are also included in **Figure 1.**

Expansion port

The main port for user add-ons is the "floppy disc" port. This has all the address, data, and control bus connections, plus a few additional lines including a +5 volt output. Connections are made to this port by way of a 2 by 25-way 0.1 inch matrix edge connector. Full connection details of this port are shown in **Figure 2.** In non-standard Z80 interfacing fashion, all sixteen address lines are used for the input/output map. The basic technique is to decode only the eight most significant address lines, but some or all of the lower eight address lines can also be decoded if more than one decoded output is needed. Addresses from &F800 to &FAFF are free for user add-ons. Full address decoding of the eight most significant address lines is not essential, and the basic method recommended by Amstrad is to have user input/output devices activated when A10 and IORQ (plus RD or WR) are low. This is similar to the method used with the Spectrum where A5 and IORQ going low are used to activate add-on circuits. Access add-ons using addresses that take the five most significant address lines high (eg addresses from —F800 to &F8FF) so that spurious operations of internal circuits are avoided. An attractive feature of the Amstrad system of interfacing in comparison with the Spectrum method is that the eight least significant address lines are not tied up by internal circuits and are available for add-ons. This conveniently enables user add-ons to be placed at consecutive addresses, and a large address range is available.

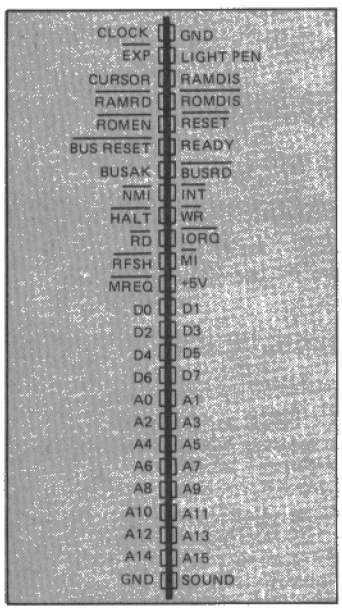


Figure 2. The "floppy disc" port

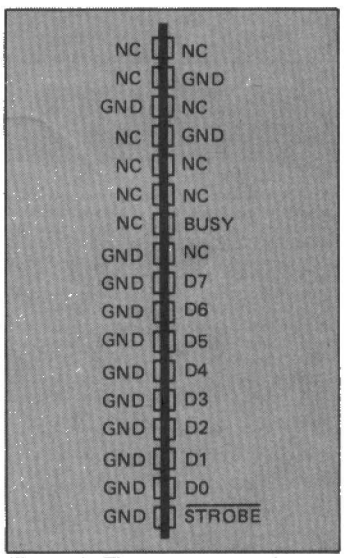


Figure 3. The port connections.

"SOUND" is an audio input, and a signal of about 50mV RMS is needed to fully drive the amplifier. A lightpen input is included, and as the CPC464 has the same video controller as the BBC machine (a 6845) a BBC compatible lightpen would presumably be suitable for the CPC464. The 6845 address register is at I/O address &8C00 while the data register is at &8D00 when writing, or &8F00 when reading.

Input and output devices can be accessed from BASIC using the normal (for Z80 based machines) OUT instruction and INP function. From machine

code they must be accessed using instructions where the B register provides the eight most significant bits of the address. Block instructions which use the P register as a counter

"The eight least significant address lines are available for add-ons".

cannot be used properly with the CPC464.

The clock signal is the 4MHz signal used for the Z80A microprocessor.

Printer Port

Details of the parallel printer port are shown in **Figure 3,** and this requires a 2 by 17 way 0.1 inch matrix edge connector. There are only seven data outputs since D7 is actually just earthed. No "Acknowledge" handshake line is provided, and the "Busy" line has to be used. Data can be written to the port at address &EF00, and the outputs latch. Bit 7 is available at the Strobe output, but this bit is inverted. The Busy line connects to bit 6 of port B of an 8255 PIA. This can be read at I/O address &F500. Obviously the printer port could be used as a general purpose output port with handshaking capability if required.

A stereo audio output is available, and this requires a 3.5mm stereo jack connector. The output level seems to be too low to drive most amplifiers.

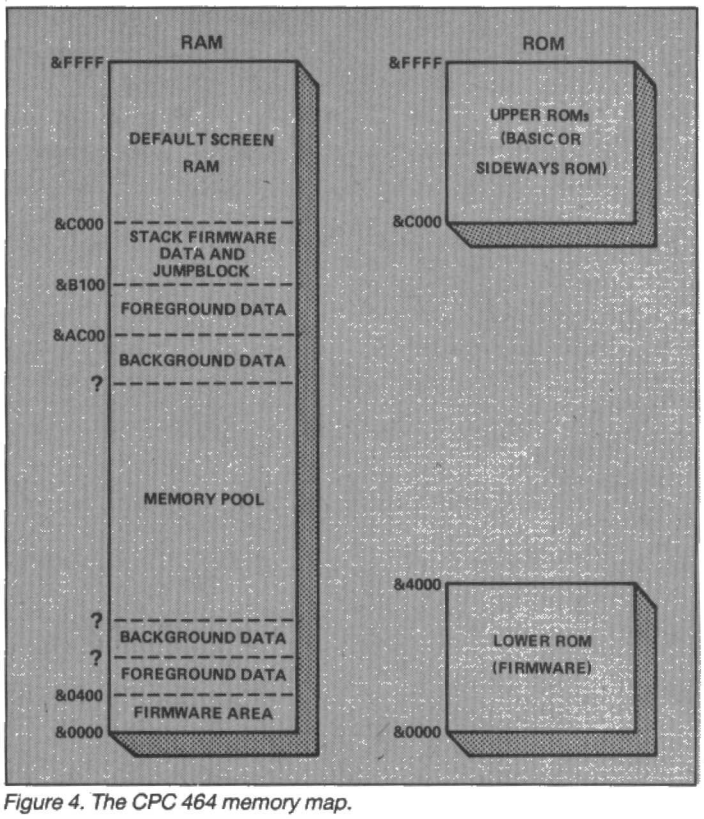


Figure 4. The CPC 464 memory map.

Memory map

Figure 4 shows the memory map for the CPC464, and this is complicated by the fact that the Z80 has a 64K address range, but the CPC464 had 64K of RAM plus 32K of ROM. The upper 16K and lower 16K of the memory map are shared by ROM and RAM; the upper ROM is either the built-in BASIC or a sideways ROM. The operating system is in the lower ROM. The firmware should be controlled by using the published routines and not by altering system variables. Many operations require several variables to be altered. The published routines do this automatically and avoid the possibility of the machine being left in an indeterminate and (perhaps) unstable state. Routines are called via a jump block in RAM. Details of addresses plus exit and entry conditions are given in the firmware manual published by Amsoft, which is essential reading for the serious CPC464 programmer.

DRAGON + FLEX – A POWERFUL COMBINATION

Lee Francis describes the way in which the flex operating system interacts with the Dragon 64 computer to provide a powerful system suitable for a wide range of applications.

For the uninitiated, Flex is an operating system for 6800 and 6809 based microcomputer systems. As such it is of interest to owners of Dragon Data's 64K computer and to BBC micro users who invest in one of the 6809 second processors that have recently been introduced for this machine. In this article we shall look at how Compusense's version of Flex performs on the Dragon computer but the information will be of equal relevance to the BBC implementation.

Flex is supplied on a single 5¼" disk that is accompanied by two manuals. One of these is a user's manual detailing the facilities provided by the Flex operating system while the other is a slimmer volume dealing with information that is specific to the Dragon implementation of Flex.

Booting up

Getting Flex up and running only requires that the system disk is placed into drive 1 (or at least the "first" drive of your system which in some cases may be referred to as drive 0 in other documentation) and the BOOT command entered from the keyboard. After a few moments of whirring and clicking from the drives, a copyright message will be displayed together with the invitation to enter the date in the form MM,DD,YY – this anglicised version of Flex still adopts the American form of day/month specification.

Having entered the date, the screen will give a very brief introduction to Flex in use followed by an invitation to see a demonstration program in action. This program illustrates the versatile windowing facilities built into Flex – these include the means by which users may configure their own windows.

When Flex is ready to accept a command line it will display a prompt that takes the form of +++. As ever the first time your master Flex disk reaches this stage the important thing to do is to produce a backup copy of the disk. To accomplish this two of the supplied utilities are called into action. The first is NEWDISK (the Flex format equivalent). In the Dragon implementation this differs from the standard Flex command in that it offers the facility to create disks of a number of different formats (single and double density, single and double sided). NEWDISK will prompt with the line:

Standard DRAgon Disk (SSDD 40 track) y/n?

unless there is a good reason this question is answered in the affirmative, at which stage the formatting process will begin. NEWDISK will ask for a name and number for the disk, these are not optional and must be supplied.

When formatting is complete the system will ask whether or not you wish to format further disks. If no further disks are to be formatted, the system will respond with the familiar +++ prompt.

The comman to backup a disk is, in a refreshingly straightforward fashion, BACKUP>; this is followed by two parameters that define the source and destination drives. In a twin disk system, assuming the master disk is in drive 0 and the freshly formatted disk in drive 1, the command would be:

BACKUP 0,1

The system also makes the provision for single disk operation in which case the usual prompts concerning disk swapping are provided.

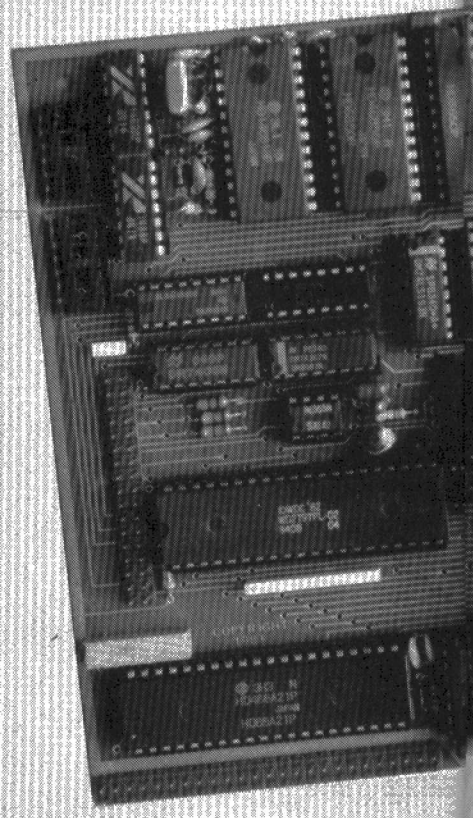
"The main function of any operating system is to provide a method of operating disk files".

Manipulating files

The main function of any operating system is to provide an orderly method of manipulating disk files, and whilst Flex can do much more, this is its bread and butter function.

Flex file names consist of three parts, the file name, an extension and a drive number. The file name is the name supplied by the user to describe the file while the extension is used by the system to specify the type of file. The drive number is again an obvious specifier but Flex treats drives in a subtle way. In a two drive

SOME ALTERNATIVE



A number of companies are now producing 6809 based processor boards that allow FLEX to be implemented on a wide variety of hardware. Cambridge Micro Systems can supply a 6809 2nd processor designed for use with the BBC micro. The board features a full complement of 64K DRAM and there is a provision for a battery back up for the on-board CMOS devices. The board connects to the BBC's tube Interface and allows standard format FLEX disks to be run on the computer. The board can be mounted inside the BBC's case or can be plugged into a Eurorack with power

system it is possible to assign one drive to be the system drive (this would usually hold the system disk) and the other drive to be the working drive. (This would hold the users disk on which text files or program data was being created.) Flex allows these drives to be specified by the ASN command which takes the form:

ASN(,W=<drive>) (,S=<drive>)

Where W and S are the system and working drive. In a twin drive system it would be usual to assign drive 0 to the system and drive 1 to the working drive with the following command:

ASN,S=0,W=1

At any stage, issuing the ASN command alone will report with the current assignment of the drives.

With the above ASN assignment Flex will assume that the first filename in a command line has the extension .CMD and that it is in drive 0. The Flex command +++CMD would thus provide a catalogue of all files with the .CMD extension on the disk in drive 0. The default assignment could be over-ridden by including an alternative drive number as part of the command line, for example +++CAT,1 would provide a list of files on the disk in drive 1.

The ASN assignment will also mean that a command such as LIST MYFILE will list the text file named MYFILE from drive 1 (the working drive) using the LIST.CMD utility in drive 0.

Users of OS9 will recognise these commands as offering similar facilities to the CHX and CHD commands.

Of the numerous facilities that Flex offers the COPY command is one of the most versatile. This can take one of three forms, the first two of which are quite straightforward and allow files to be transferred between drives, either keeping their original name or having a new name specified for the copied version. The third option takes the form:

COPY,<drive>,<,drive>
(,<match list>)

This is the most powerful and versatile form of the command as it allows a certain group of files (those that match the characters given in the match list) to be copied between drives.

Examples of this option would be:

+++COPY,0,1
+++COPY,1,0,.CMD,.SYS
+++COPY,0,1,A,B,CA.T

The first is equivalent to the BACKUP command and would copy all files from drive 0 to drive 1. The second will copy those files on drive 1 with the extensions .CMD and .SYS to drive 0, while the third will copy those files on drive 0 that start with either A or B (regardless of extension) and those files that start with CA together with all files with the extension .T.

It can be seen that the Flex COPY command is one of the most versatile found in any OS and would be a welcome addition to the command set of some other well known operating systems.

Flex provides a host of other commands that are fully documented in the accompying manuals including the CS utility that allows one of eight different character sets to be selected as well as allowing the user to create their own set for special applications.

The above has, it is hoped, given some idea of the powerful facilities of the Flex operating system. Flex and OS9 fight it out for the affections of the users of 6809-based micro systems although many would argue that Flex is more suited to small microcomputer systems than is OS9. This is by virtue of the fact that Flex makes no attempt to offer the multi-tasking and multi-user options of OS9. While these are desirable features in larger systems, they are, it is argued, unused complexities on systems such as the Dragon.

In choosing Flex as an operating system, the user will have at their disposal one of the most powerful eight bit operating systems and one that has a wealth of applications software written for it – all available off the shelf, now.

"Flex is one of the most powerful eight bit operating systems – one that has a wealth of applications software written for it".

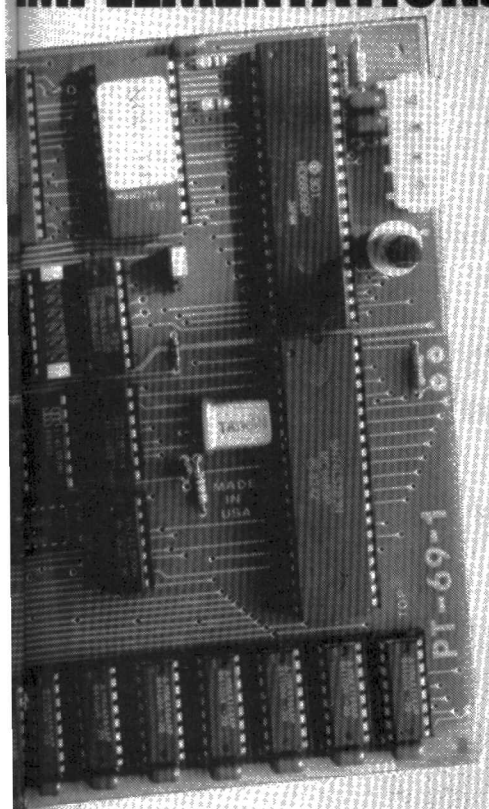
IMPLEMENTATIONS

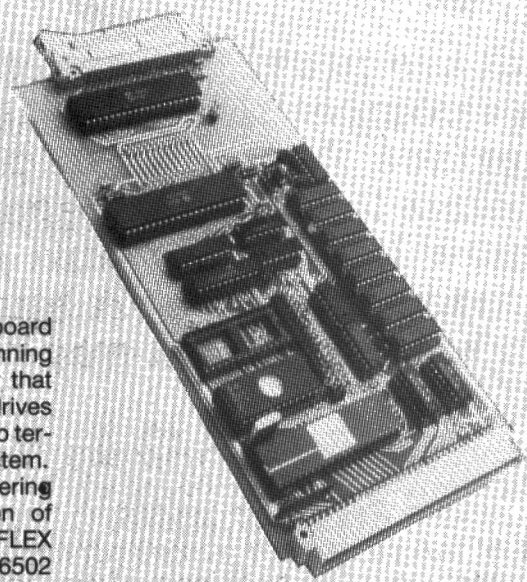
All of these products provide an ideal basis for the development of stand alone computer control systems, the power and versatility of the FLEX OS contributing greatly to reduced development times.

Cambridge Micro Systems
44a Hobson Street
Cambridge
CB1 1NL
0223 324141

Compusense
286D Green Lanes
Palmers Green
London
N1 35XA
01-882 0681

Ralph Allen Engineering
Forncett End
Norwich
Norfolk
095 389 420





supply and backplane.

Compusense also supply a single board 6809 computer that is capable of running FLEX. This is a complete computer that only requires the addition of disk drives (the controller is on board) and a dumb terminal to form a fully fledged FLEX system.

A 6809 card from Ralph Engineering offers Tangerine owners the option of upgrading their computer to full FLEX capability. This card replaces the 6502 card of the standard system and provides a true FLEX system rather than a terminal talking to a 6809 card.

SPECTRUM WORD PROCESSOR

Alan D. Went describes a versatile text processor for the Spectrum computer.

ZX Words, is a Simple Word Processing program for the Spectrum, allowing text of up to approx 11,000 characters (about 3 A4 pages) top be entered, edited and printed to screen, ZX Printer, or, with an appropriate interface, a full size Printer. The text can also be saved and reloaded as required.

The program should be entered as listed, omitting all REM's, and save the program before Running. To Start ZX Words RUN 9500, this will set up graphics for cursors and poke the machine code break disable routine into place. I suggest that, until you have checked that your version works, you change Line 3770 to:- 9770 REM RANDOMISE USR 65110, deleting the REM only when you are happy that everything is OK. Your final Version should be saved by GOTO 9999, this will ensure that on loading ZX Words will run automatically.

Instructions for use

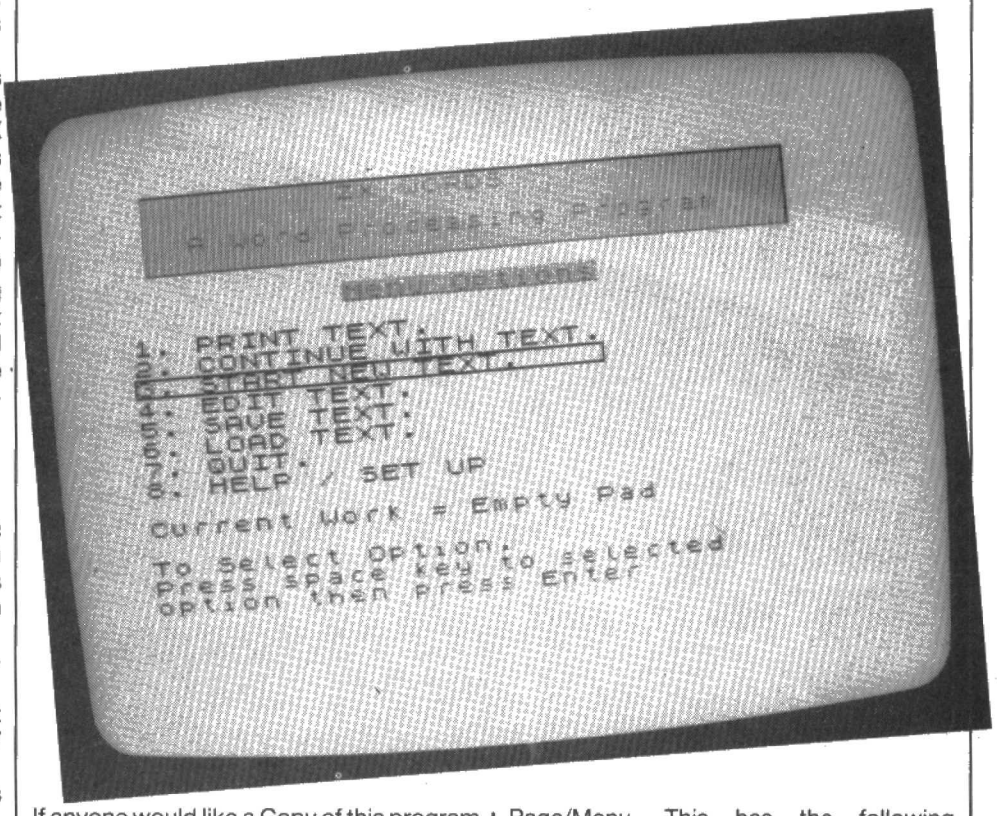
On Loading you will be presented with a Menu, Option 1 of which will be enclosed in a bright box. To select an option this box is moved down with the space key and an option selected with the Enter Key.

Option 8 will show all commands available with RUN Words. except:-
In Print Mode: holding 'S' will pause print at the end of the current Line; 'C' will COPY page to ZX Printer; 'Q' will QUIT printing.

Note. To exit from Entry Mode use Caps Shift + 3.

Program structure

Text is entered and held in a single dimension array A$(Len) where Len is selectable according to the maximum length of text to be entered. (default = 9000). Characters are typed straight onto the screen via an IN KEY$ routine.



If anyone would like a Copy of this program and does not wish to type it in I will be happy to supply a copy on cassette for £4.00 (inc p&p) from *25 Lucy Close, Stanway, Colchester.*

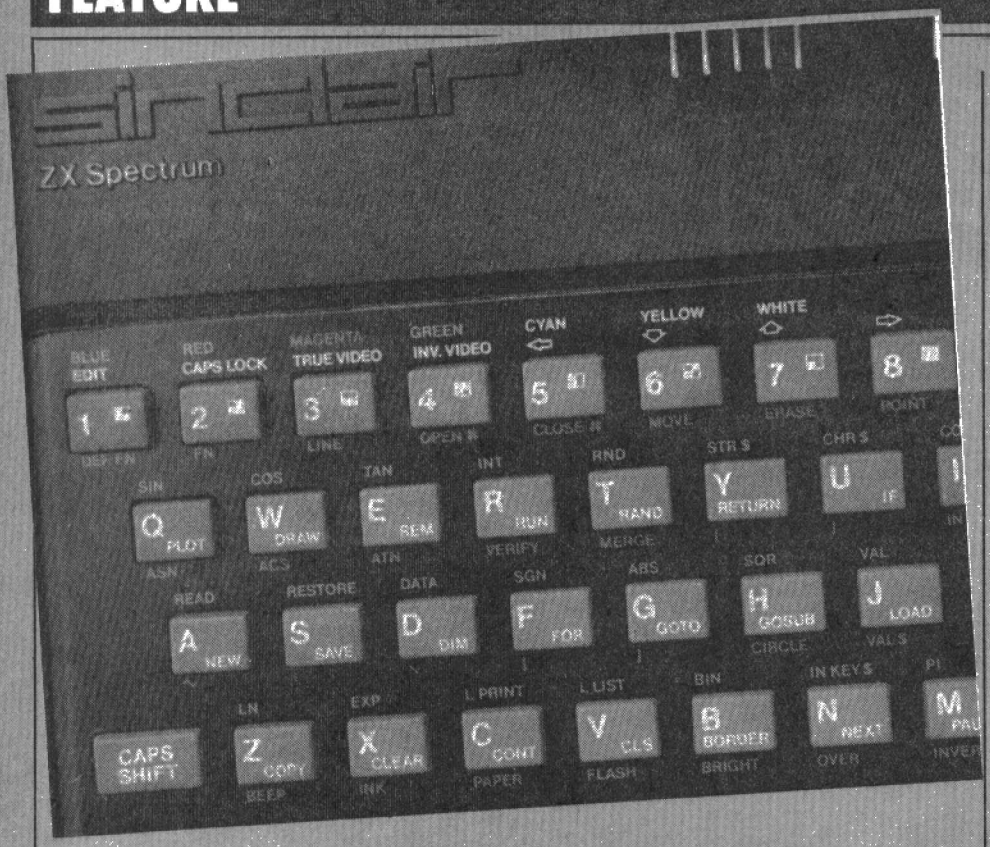
Loading instructions

LOAD ""
On loading you are presented with a Title Page/Menu. This has the following options:-

1. Print Text.
2. Continue with Text.
3. Start new Text.
4. Edit Text.
5. Save Text.
6. Load Text.
7. Quit.
8. Help/Set up.

Location of ZX Word's major routines.

Line	
100/110	Capital Lock, On/Off.
130/140	General Purpose Single Character Input routine.
150	Print Page of text with Edit Cursor.
120	Print Page without Cursors.
1000 to 1030	Main Text Input Loop.
1040 to 1099	Commands during Entry.
2000 to 2180	Instruction Pages (3)
2200 to 2600	Rename Text – Set Line Length – Set Text Length option selected by bright cursor.
3000 to 3070	MAIN MENU and Option Selection.
4500 to 4530	Save and Verify.
4700 to 4720	Load Text. Change these two routines if Using Microdrive for text store
4900 to 4999	Quit.
5000 to 6100	Edit Mode Cursor and Command Loop
6100 to 7110	Various Editing Sub Routines.
8000 to 8190	Print Text. Prints text to Screen or Printer.
9000 to 9010	Clear Text and Reset Variables.
9500 to 9550	Set up Graphics for Cursors.
9700 to 9770	Break Disable Routine. This is an interrupt driven routine that checks for Break keys and if pressed forces a CONTINE to be executed. This routine could be used in any Basic program To turn Break Disable ON use RAND USR 65110 To turn Break Disable OFF use RAND USR 65120



Option 1 will be in a bright box, this is the Option Cursor, to select an item from the list use the SPACE key to move the Cursor down the list, then, when on the correct option press ENTER.

When the program is first loaded no text will be present.

On side 2 of the tape, the program contains an example text; it is suggested that this is loaded when the program is used for the first time.

Options

1. Print Text

This will allow the text to be printed, either to the screen or to a printer. For use with a printer other than the Sinclair please refer to the notes at the end of these instructions.

To the question **Printer ? [Y or N] 'Y'** will turn on the printer, any other input will select the screen.

To the Question, **Characters per line?** the current number is show and pressing ENTER will leave this unchanged. To change this type number then ENTER. The ZX Printer will print a maximum of 32 characters per line, but shorter lines can be chosen. For other printers much longer lines can be used.

The text will then be printed as required, at the end touching any key will return to the menu. During printing holding down the 'S' key will pause the printing at the end of the next line.

The 'C' key will copy the screen to the printer, and clear the screen.

The 'Q' key will return to the menu.

2. Continue with Text

This is the main text entry option, and also allows more text to be added to an existing document. A display of space left is shown at top Right Corner of the screen.

A 'C' in the top Left Corner indicates that CAPS LOCK is ON.

The next letter will be typed in the position indicated by a '*'.

During entry of text some of the Number keys when used with the CAPS SHIFT key have different functions as below.

1 = Select EDIT MODE.
2 = Caps Lock ON/OFF.
3 = Return to MAIN MENU.
4 = Turn BACK 1 PAGE.
9 = Turn FORWARD 1 PAGE.
0 = CORRECT last Character. (Will repeat if held down.)

ENTER will give a definite New Line. To indent a paragraph during printing leave 3 spaces after using ENTER. The position of an ENTER Character on the display is indicated by a small dot.

To add to an existing text use CAPS SHIFT + 9 to locate the * and then start typing.

3. Start New Text

This will clear the existing document and ask for a name for a new one. (Max. 10 letters). Then you are returned to the menu.

4. Edit Text

To edit your text this mode is selected either from the Menu or by Caps Shift + 3 from Option 2 Text Entry.

The cursor is moved to the point to be edited with the following keys (No SHIFT)

2 = To middle line.
3 = To page end.
4 = Turn back 1 page.
5 to 8 = As arrows above keys.
9 = Turn forward one page.
0 = To start of page.

or 5 (Search) then type incorrect word or part of word ENTER, a search is then made from the current cursor point for those letters.

Note. The cursor is marked by a box around that letter.

When the cursor is at the required point then the following options may be selected by touching key:-

C = Change the letter in the cursor. The cursor will advance 1 character after each change. To return to cursor move press ENTER.

I = Insert. To insert a letter(s) 1 Character behind the cursor, ENTER to end.

P = Insert paragraph marker. To force a new line during printing.

D = Delete, then L deletes the letter in the cursor. W deletes the word starting at the cursor. S deletes the sentence (to the next'.') storing this for later use. Again ENTER returns you to cursor movement.

L = Look at stored sentence.

R = Replace sentence starting at cursor position, moving all the text up to make room.

X = return to either menu or Text entry.

5. Save Text

This will save the document to tape with name given in option 3 or 8.

6. Load Text

To reload previous work, name of work can be a name given to a document on tape or just ENTER to load the next document on the tape sequence.

7. Quit

Ends use of Program.

8. Help/Set up

This gives 3 pages of information on commands available, then the option to change the name of the current document, Line length, or max Text length (with effect from next new document).

To change program for a different printer, line 8000 will need to be changed to select between screen and printer, and Line 8140 to send text to printer.

```
    1 REM ****************************
    2 REM
    3 REM ZX WORDS FOR SPECTRUM 48K
    4 REM
    5 REM ****************************

   12 REM When Program first run GOTO 9500 THEN USE GOTO 3000
   99 REM Caps Lock
  100 IF PEEK 23658)=8 THEN  POKE 23658,(PEEK 23658)-8: PRINT AT 0,0;" ": B
EEP .2,2: RETURN
  110 POKE 23658,8: PRINT AT 0,0; PAPER 1: INK 6;"C": BEEP .2,10: RETURN
  129 REM Single Letter Input Routine
  130 LET I$=INKEY$: LET i=CODE i$: IF I$="" THEN  GO TO 130
  131 BEEP .01,15: RETURN
  149 REM Print Page with Cursor (for Edit)
  150 LET B$(CUR1)="_Graphic A": PRINT AT 1,0;A$(PAGE-639 TO PAGE);AT 1,0;
OVER 1;B$: LET B$(CUR1)=" ": RETURN
  170 PRINT AT 1,0;a$(page-639 TO page): RETURN
 1000 REM DISPLAY AND INPUT
 1005 CLS : POKE 23658,0
 1010 PRINT &1;AT 1,0; PAPER 4; INK 9;"ENTRY MODE                    "; P
RINT AT 0,3;"Page ";page/640;AT 0,14;"Space Left =
 1015 REM MAIN LOOP
 1020 LET a$(cur)="*": PRINT AT 1,0;A$(PAGE-639 TO PAGE);AT 0,26;STR$ (LEN-
CUR+1)+" "
 1030 LET I$=INKEY$: LET i=CODE i$: IF I$="" THEN  GO TO 1030
```

```
1035 BEEP .01,10: IF I>127 OR I<32 THEN  GO TO 1040
1036 LET A$(CUR)=I$: LET CUR=CUR+1: IF CUR>=PAGE THEN  LET PAGE=PAGE+320:
 PRINT AT 0,8;INT (.5+(page/640));"  ": IF page>=len THEN  PRINT &1;AT 1,0;
 FLASH 1;"     NO MORE ROOM                ": PAUSE 500: RETURN
1038 GO TO 1020
1039 REM COMMAND CHECKING.
1040 LET A$(CUR)=" ": IF I=6 THEN  GO SUB 100: GO TO 1030
1050 IF I=12 THEN  LET a$(cur-1 TO )="  ": LET cur=cur-1: GO TO 1020
1060 IF I=13 THEN  LET A$(CUR)=" Graphic B": LET CUR=INT ((CUR-1)/32)*32+3
3: IF CUR>=PAGE THEN  LET PAGE=PAGE+320: PRINT AT 0,8;INT (.5+(page/640));
"  ": IF page>len THEN  LET page=len
1061 IF cur>=len THEN  PRINT &1; FLASH 1;AT 1,0;"NO MORE ROOM": PAUSE 500:
 RETURN
1065 IF I=13 THEN  GO TO 1020
1070 IF I=7 THEN  LET A$(CUR)=" ": GO SUB 5000: GO TO 1010
1080 IF I=4 THEN  BEEP 1,-10: RETURN
1090 IF I=5 THEN  BEEP 1,0: LET page=page+640: IF page<=640 THEN  LET page
=640
1095 IF I=15 THEN  BEEP 1,0: LET page=page+640: IF page>=len THEN  LET pag
e=len
1096 PRINT AT 0,8;page/640;"  "
1099 BEEP .2,-20: GO TO 1020
1999 REM HELP PAGES AND SETTINGS
2000 CLS : PRINT TAB 10;"ZX WORDS"'' BRIGHT 1;"Commands from Entry Mode"
2010 PRINT "CAPS SHIFT + "'"1 = Goto Edit Mode" "2 = Caps Lock ON/OFF"'"3
 = Return to Main Menu" "4 = Back 1 page" "9 = Forward 1 page"'"0 = Correc
t last Character"''          **********  "ENTER = New Line / Paragraph
(To indent paragraph leave      3 spaces at start of line       following
a ENTER.)"
2020 PAUSE 30: PRINT &1;AT 1,0;"Press a key to continue": IF INKEY$="" AND
 IN 65278=255 AND IN 32766=255 THEN  GO TO 2020
2030 LET a=2: LET b=12: CLS : PRINT  BRIGHT 1;"Commands from EDIT MODE"
2040 PRINT AT 1,0;"KEY MODE     ACTION REQUIRED";AT 1,0; OVER 1;"--- ----
---------------"
2050 PRINT "C";TAB a;"Change";TAB b;"""Enter " = End"
2060 PRINT "D";TAB a;"Delete";TAB b;"L = Single Letter";TAB b;"W = Word";
TAB b;"S = Sentance(Stored";TAB b;"for later use)";TAB b;"isee Replace)";T
AB b;"""Enter"" = End"
2070 PRINT "I";TAB a;"Insert";TAB b;"""Enter"" = End"
2080 PRINT "L";TAB a;"Look";TAB b;"at stored sentance"
2090 PRINT "P";TAB a;"Paragraph ";TAB b;"insert Para. Mark. Graphic B"
2100 PRINT "R";TAB a;"Replace";TAB b;"Insert Sentance"
2110 PRINT "S";TAB a;"Search";TAB b;"word then ""Enter"
2120 PRINT "X";TAB a;"Return";TAB b;"Back to Entry Mode"
2130 PAUSE 20: PRINT &1;AT 1,0;"Press a key to continue": IF INKEY$=""  AND
 IN 65278=255 AND IN 32766=255 THEN  GO TO 2130
2140 CLS : PRINT  BRIGHT 1;"EDIT MENU     CURSOR MOVES"
2150 PRINT  "KEY    ACTION"AT 2,0; OVER 1;"---  ------"
2160 PRINT "2 =  To Middle Line    3 =  To Page End    4 =  Back 1 Page
"5-8   As Arrows"  9 =  Forward 1 Page"  0 =  To Page Start
2170 PRINT "(Note NO SHIFT KEYS Used)"
2180 PAUSE 30: PRINT &1;AT 1,0;"Press a key to continue": IF INKEY$=""  AND
 IN 65278=255 AND IN 32766=255 THEN  GO TO 2180
2200 REM SET UP
2220 CLS : LET b=30: PRINT  BRIGHT 1;"SET UP DETAILS
2225 FOR i=1 TO 12 STEP 3
2230 PRINT AT 3,0; "No Change"    Text Name      ";TAB b;a    "Line Length
   =";TAB b;cha    "Max Text Length    (Max 12000)";TAB b;ser
2235 PRINT  ""To Select Option," "Press space key to selected      option
then press Enter"
2240 PRINT  OVER 1; BRIGHT 1;AT i+2,0; Graphic C Graphic D Graphic B Graph
ic D Graphic D Graphic D Graphic D Graphic D Graphic D Graphic D Graphic
D Graphic D Graphic D Graphic D Graphic D Graphic E"
2250 IF INKEY$="" THEN  GO TO 2250
2260 IF INKEY$="  " THEN  PRINT AT i+2,0;"                    ": NEXT i: GO TO 2
225
2270 IF INKEY$=CHR$ 13 THEN  GO TO 2290
2280 GO TO 2250
2290 RESTORE 2300: FOR a=1 TO 3 STEP 3: READ c: NEXT a: GO TO c
2300 DATA 2310,2400,2500,2600
2310 RETURN
2400 INPUT "Enter New Name (Max 10 Letters):";x$: IF LEN x$>10 THEN  LET x$
=x$( TO 10)
2405 PRINT AT 3,b;"
2410 GO TO 2200
2500 INPUT "New Line Length (ZX Printer = 32) ";cha: DIM n$(cha+2): GO TO
2200
2600 INPUT "New Text Length (will apply next time work is cleared) ";ser
: GO TO 2200
2999 STOP
3000 REM MAIN MENU
3001 CLS : PRINT AT 1,10;"ZX WORDS "''"    A Word Processing Program
3005 PLOT 8,168: DRAW 240,0: DRAW 0,-32: DRAW -240,0: DRAW 0,32
3006 FOR a=1 TO 4: PRINT  OVER 1; INK 2; PAPER 6; BRIGHT 1;AT a,1:
                    ": NEXT a
3010 PRINT AT 6,10; PAPER 6; INK 1;"Menu Options"; OVER 1;AT 6,10; ------
-------
3019 FOR I=1 TO 8:
3020 PRINT AT 8,0;"1. PRINT TEXT.""2. CONTINUE WITH TEXT.    3. START NEW
TEXT."
3030 PRINT "4. EDIT TEXT.""5. SAVE TEXT." "6. LOAD TEXT." "7. QUIT.    8.
HELP / SET UP"
3035 PRINT ""Current Work = ";x$
3039 PRINT ""To Select Option.""Press space key to be selected      option t
en press Enter"
3041 PRINT AT 7+I,0; BRIGHT 1; OVER 1; Graphic C Graphic D Graphic B Graph
ic D Graphic D Graphic D Graphic D Graphic D Graphic D Graphic D Graphic
D Graphic D Graphic D Graphic D Graphic D Graphic D Graphic E
3042 IF INKEY$="" THEN  GO TO 3042
3043 IF INKEY$=" " THEN  PRINT AT 7+I,0;"
: NEXT I: GO TO 3019
3045 IF INKEY$=CHR$ 13 THEN  GO TO 3049
3046 GO TO 3042
3049 BEEP .5,12: CLS
3050 RESTORE 3050: FOR A=1 TO 1: READ B: NEXT A: GO SUB B: GO TO 3000
3060 DATA 9000,1000,3000,5000,4500,4700,4900,2000
3070 GO TO 3000
4500 REM SAVE A$
4510 LET a$(len-9 TO len)=x$: LET a$(len-10 TO len-10)=STR$ cur: SAVE x$ D
ATA a$()
4520 PRINT &1;AT 1,0;"To Verify press ""v"" (other=skip)": GO SUB 130: IF
 i$="V" OR i$="v" THEN  PRINT "Start Tape Playing NOW." "If error occurs T
ry Goto 4510  to Re-record.": VERIFY x$ DATA a$()
4530 RETURN
4700 REM Load A$
4710 INPUT  Name of Work to be loaded? ";x$
4720 CLS : PRINT "Play Tape NOW"'' If Load Error press CONT and     Play Tap
e again.": LOAD x$ DATA a$(): LET len=LEN a$: LET x$=a$(len-9 TO len): LET
 cur=VAL a$(len-16 TO len-10): GO TO 3010
4900 CLS : PRINT ''"ZX WORDS"'' ''Spectrum WORD PROCESSOR '''"© ALAN D. WE
NT." "28 LUCY CLOSE" "STANWAY" "COLCHESTER" "ESSEX"
4999 STOP
5000 REM EDIT ROUTINES
5010 CLS : DIM B$(640): LET PAGE=640: LET CUR=1
5020 GO SUB 110: PRINT &1;AT 1,0; PAPER 2; INK 7;"EDIT MODE";TAB 21;"CURSO
R MOVE"
5030 IF CUR<1 THEN  LET CUR=1
5040 IF CUR1>PAGE THEN  LET PAGE=PAGE+640: LET CUR1=CUR1-640
5045 GO SUB 150
5050 LET I$=INKEY$: IF I$="" THEN  GO TO 5050
5055 LET I=CODE I$-48: IF I<0 OR I>10 THEN  GO TO 5150
5060 LET CUR1=CUR1+(I=8)-(I=5): IF I=5 OR I=8 THEN  GO TO 5030
5070 IF I=6 THEN  LET CUR1=CUR1+32: GO TO 5030
5075 IF I=7 THEN  LET CUR1=CUR1-32: GO TO 5030
5080 IF I=3 THEN  LET CUR1=640: GO TO 5030
5100 IF I=2 THEN  LET CUR1=321: GO TO 5030
5110 IF I=0 THEN  LET CUR1=1: GO TO 5030
5120 IF I=9 THEN  LET PAGE=PAGE+640: LET CUR1=1
5130 IF I=4 THEN  LET PAGE=PAGE-640: LET CUR1=1
5140 IF PAGE<640 THEN  LET PAGE=640
5150 IF cur>len THEN  LET cur=len
5155 IF I$="X" THEN  BEEP .2,1: RETURN
6000 LET cur2=cur1+(page-640)
6020 IF I$="P" THEN  BEEP .2,(CODE i$-60): GO TO 7100
6030 IF I$="S" THEN  BEEP .2,(CODE i$-60): GO TO 7000
6040 IF I$="L" THEN  BEEP .2,(CODE i$-60): GO TO 6300
6050 IF I$="C" THEN  BEEP .2,(CODE i$-60): GO TO 6400
6070 IF I$="R" THEN  BEEP .2,(CODE i$-60): GO TO 6600
6080 IF I$="D" THEN  BEEP .2,(CODE i$-60): GO TO 6700
6090 IF I$="I" THEN  BEEP .2,(CODE i$-60): GO TO 6900
6100 BEEP .5,-40: GO TO 5020
6300 REM INSERT
6310 GO SUB 100: PRINT &1;AT 1,14; PAPER 5;"INSERT ""ENTER=END"""
6320 GO SUB 150
6324 GO SUB 130: IF I=13 THEN  GO SUB 110: GO TO 5020
6325 IF I=6 THEN  GO SUB 100: GO TO 6324
6330 LET A$(CUR2+1 TO CUR+1)=A$(CUR2 TO CUR): LET A$(CUR2)=I$: LET CUR2=CU
R2+1: LET CUR=CUR+1: LET CUR1=CUR1+1: GO TO 6320
6400 REM CHANGE
6410 GO SUB 100: PRINT &1;AT 1,14; PAPER 5;"CHANGE ""ENTER=END"""
6420 GO SUB 150: GO SUB 130: IF I=13 THEN  GO SUB 110: GO TO 5020
6430 IF I=6 THEN  GO SUB 100: GO TO 6420
6450 LET A$(CUR2)=I$: LET CUR2=CUR2+1: LET CUR1=CUR1+1: GO TO 6420
6600 PRINT &1;AT 1,15; INK 6;"REPLACE": LET A$(CUR2+LEN M$ TO CUR+LEN M$)=
A$(CUR2 TO CUR): LET A$(CUR2 TO CUR2+LEN M$-1)=M$: LET CUR=CUR+LEN M$:
6610 GO TO 5020
6700 REM delete
6710 PRINT &1;AT 1,14; PAPER 5;"DELETE (L, W OR S)"
6720 GO SUB 150: GO SUB 130
6730 IF I$="L" THEN  LET A$(CUR2 TO CUR)=A$(CUR2+1 TO CUR+1): LET CUR1=CUR1
-1: LET CUR=CUR-1: GO TO 6720
6740 IF I$="W" THEN  GO TO 6800
6750 IF A$(CUR2)=" "  THEN  LET A$(CUR2 TO CUR)=A$(CUR2+1 TO CUR+1): LET CU
R=CUR-1: GO TO 6750
6760 GO TO 6700
6800 IF I$="S" THEN  GO TO 5020
6810 FOR A=CUR2 TO CUR: IF A$(A)=".", THEN  GO TO 6825
6820 NEXT A: PRINT &1;AT 1,14; FLASH 1; "No "."" FOUND       ": PAUSE 150:
GO TO 6700
6825 IF a$(A+1)=" Graphic B" THEN  LET a=a+1
6830 LET A=A+1: LET M$=A$(CUR2 TO A): LET A$(CUR2 TO CUR)=A$(A TO CUR+(A-
CUR2))
6840 LET cur=cur-LEN m$+1: GO TO 6700
6900 CLS : PRINT "Stored Text for Replacement :-    ""  ";m$;" "
6910 PAUSE 250: CLS : GO TO 5020
7000 REM SEARCH
7010 GO SUB 100: INPUT  PAPER 5;"SEARCH (ENTER WORDS)";i$: LET i=LEN i$-1
7020 FOR A=CUR2+1 TO CUR: IF A$(A TO A+1)=i$ THEN  LET CUR1=A: GO TO 5040
7030 NEXT A: GO TO 5020
7040 IF A<640 THEN  LET CUR1=A: GO TO 5020
7050 LET A=A-640: GO TO 7040
7100 LET i$=" Graphic B"
7110 LET C$=A$(CUR2 TO CUR): LET A$(CUR2)=i$: LET CUR2=CUR2+1: LET CUR=CUR
+1: LET CUR1=CUR1+1: LET A$(CUR2 TO CUR)=C$: GO TO 5030
8000 PRINT &1;"Printer? (Y or N)                ": GO SUB 110: GO SUB 100: L
LT PR=2+(i$="Y")
8002 CLS : PRINT "Number of characters per line   (ZX Printer ma   )
";cha;"To change enter new number     Press  enter  only to leave ": INPUT
 LINE i$: IF i$<>"" THEN  LET cha=VAL i$
8005 CLS : PRINT "To print from the start of a "'"page rather than by Text
 page No then ENTER" "Press ENTER only for print from Start": INPUT  LINE
i$: IF i$="" THEN  GO TO 8010
8006 LET i=(VAL i$-1)*640: IF i>len OR i<640 OR i>CUR THEN  GO TO 8005
8007 LET a=i: CLS : LET p=0: GO TO 8030
8010 CLS
8020 LET P=0: LET A=1: REM FIND FIRST CR
8030 LET ST=A
8035 IF A$(ST)<>" " THEN  GO TO 8070
8040 LET ST=ST+1: IF ST=A+CHA-1 THEN  LET ST=A: GO TO 8070: REM EMPTY LINE
8050 IF A$(ST)=" " THEN  GO TO 8040
8060 IF P=1 THEN  LET P=0: IF A$(ST-3 TO ST-1)="     THEN  LET ST=ST-
8065 REM FIND LAST, 1ST CHECK FOR PARAGRAPH MARKER
8070 FOR B=ST+CHA-1 TO ST STEP -1: IF A$(B)=" Graphic B" THEN  LET END=B:
LET P=1: GO TO 8100
8080 NEXT B: LET END=ST+CHA-1
8090 IF A$(END)=" " OR A$(END+1)=" " OR A$(END)="," OR a$(END)=" Graphic B
" OR A$(END)="," OR A$(END)="?" OR A$(END)="!" OR A$(END)=".  OR A$(END)="
;" THEN  GO TO 8100
8092 LET END=END-1: IF A$(END)=" " THEN  GO TO 8100
8094 IF END<ST THEN  GO TO 8092
8096 LET ST=A+CHA-1
8100 LET I=0: IF A$(END)=" Graphic B" THEN  LET END=END-1: LET I=1
8120 REM IF DIFFERENT PRINTER   USED CHANGE LINE 8140 TO SUIT   NEW OUTPUT
 TYPE OR CHANGE PR     TO OUTPUT TO A DIFFERENT DATA   STREAM

8140 PRINT #pr;a$(ST TO END)
8145 IF PR=2 THEN  POKE 23692,255
8146 IF INKEY$="C" THEN  COPY : CLS
8147 IF INKEY$="S" THEN  GO TO 8147
8148 IF INKEY$="0" THEN  GO TO 8190
8150 LET A=END+1+I: IF A<CUR THEN  GO TO 8030
8160 IF pr=2 THEN  PRINT &1;"Press a key to continue
8165 IF PR=3 THEN  GO TO 8190
8170 IF INKEY$="" AND IN 65278=255 AND IN 32766=255 THEN  GO TO 8170
8180 DIM N$(CHA+1): RETURN
9000 CLS : INPUT "Name for this piece of work       (Max. 10 Letters)";x$:
IF LEN x$>10 THEN  LET x$=x$( TO 10)
9005 LET len=INT (len/640)*640+640: LET CUR=1: DIM A$(LEN+25):
9010 LET cha=32: DIM n$(cha): LET m$=" ": LET pag=640: RETURN
9500 RESTORE 9500: FOR A=0 TO 33: READ B: POKE USR "A"+A,B: NEXT A
9510 DATA 255,129,129,129,129,129,129,255: REM GRAPHICS A= Graphic A
9520 DATA 0,0,0,0,0,0,0,128: REM GRAPHICS B= Graphic B
9530 DATA 255,128,128,128,128,128,128,255: REM GRAPHICS C= Graphic C
9540 DATA 255,0,0,0,0,0,0,255: REM GRAPHICS D= Graphic D
9550 DATA 255,1,1,1,1,1,1,255: REM GRAPHICS E= Graphic E
9600 CLEAR 65109: RESTORE 9600: PRINT "Please Wait"
9710 FOR A=65110 TO 65128: READ B: POKE A,B: NEXT A
9720 READ C: LET C=C+65128: FOR A=65129 TO C: READ B: POKE A,B: NEXT A
9730 DATA 62,9,237,71,237,94,201
9740 DATA 0,0,0,62,62,237,71,237,86,201,0,0
9750 DATA 56,245,58,58,92,254,20,40,4,241,195,56,0,205,176,22,42,89,92,205
,82,22,35,62,232,119,241,241,253,52,13,1,3,0,17,112,92,33,68,92,253,203,10
,126,40,1,9,237,184,253,54,10,255,251,195,244,18: REM break disable
9770 RANDOMIZE USR 65110
9800 LET x$="Empty Pad": LET len=9000: GO SUB 9005: GO TO 3000
9900 REM ************************
9960 REM copyright ALAN D. WENT 1984
9980 REM ************************
9999 CLEAR : SAVE "ZX Words" LINE 9500: STOP
```

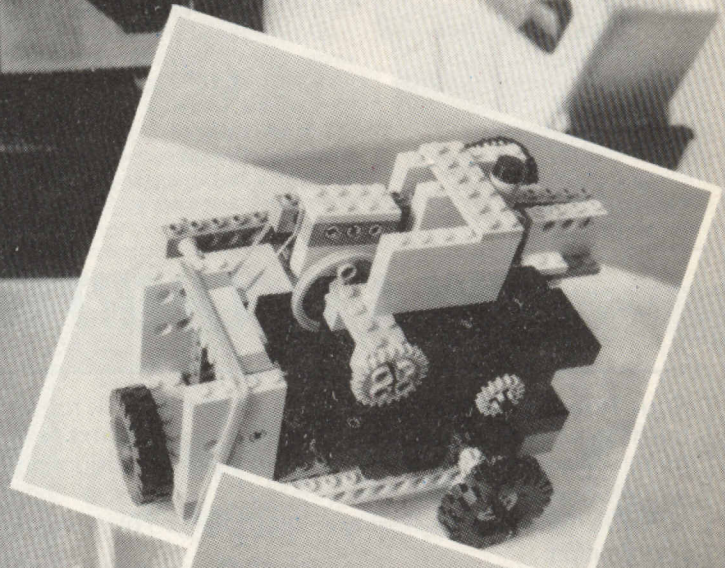
your ROBOT

An EMAP Publication

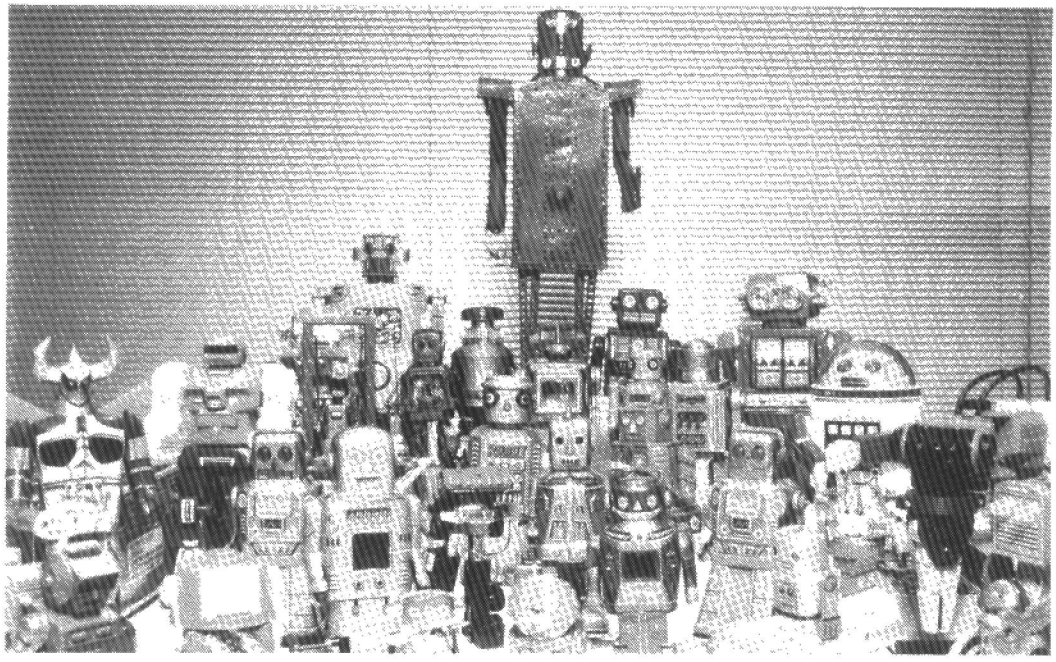
BRITAIN'S FIRST ROBOTICS MAGAZINE

OCTOBER 1984

PILOT ONE'S INTERFACE REVIEWED



PLUS: Building low cost Turtles



ROBOTS PAST, PRESENT AND FUTURE AT THE VICTORIA & ALBERT MUSEUM

Until the 25th of October, the Boilerhouse at London's Victoria and Albert museum has been taken over by a collection of robots. The blurb handed out at the press viewing stated that there were two themes running through the exhibition. The first was an exploration of 'robo-fear' during the 20th century with set pieces to illustrate the many forms in which this fear can manifest itself. The second, more practical aspect of the exhibition was to demonstrate some of the potential of today's industrial robots.

For most **Your Robot** *readers the exhibition is likely to be rather disappointing. While the V&A have done a superb job in presenting the various exhibits, the material on display is itself not of great interest. There are the odd exceptions, the collection of toy robots spanning the past three decades will revive many memories and some of the early computer hardware on display is equally intriguing. As for the industrial robots going through their set piece routines, many people will have seen it all before, if not 'in the flesh' then at least in the Fiat TV commercial.*

It is a shame that the V&A have missed the chance to stage the sort of show that would seek to promote the cause of robotics in a more positive fashion. The only domestic/educational robots on display are Hero and RB5X – both designs that originated outside this country. The wealth of low cost devices designed and produced in this country are not represented.

The exhibition has though spawned an excellent publication in the form of the guide. This is simply called Robots and while it follows the general theme of the exhibit itself, the exercise seems far more effective in the printed form. Anyone interested in the history of robots and a little 'future think' should make sure they read a copy of the book. It can be obtained from the Victoria and Albert Museum, London SW7 2RL at a cost of £2. Add 80p p&p if you are ordering by post.

If you are in the area of the V&A then do take the odd few minutes to wander round the exhibition, for most people, however, it is not going to be worth making a special trip – buy the book instead.

Robots
The Boilerhouse
Victoria and Albert Museum
London
Until October 25th
Admission free

WORKSHOP FOR ROBOT BUILDERS

From the first of this month, 121 Ifield Road, London, SW10, has provided a showcase for the numerous small robots used in education and training.

The workshop is sponsored and funded by The Entryphone Company and their managing director, Gordon Ashbee, hopes that the workshop will become a meeting place for those interested in low cost robotics equipment. He is particularly keen to talk to anyone experimenting with robots or related equipment.

The workshop can be reached on 01-373 8571.

SPECTRUM CONTROL BOX

Datel electronics have recently launched a control interface for the Spectrum under the name of RoboTEK. As the name suggests the company see robot builders as a prime market for the product.

The device provides four output channels which are simply toggled by an OUT instruction to port 63. The output consists of a changeover relay rated at 50V 1Amp.

Eight inputs are also provided. These consist of standard TTL lines, the state of which can be read by an IN instruction to port 63 once again.

The interface plugs into the Spectrum's user port while connections to the I/O lines are made via an IDC plug mounted on the top of the unit – a ribbon cable terminating in a suitable connector is supplied with the RoboTEK.

The instructions supplied with the unit are brief and to the point, they do however contain all the information necessary to make use of the unit.

The RoboTEK retails at £29.99 inclusive of post and packing and can be obtained from Datel at Unit 8, Fenton Industrial Estate, Dewsbury Road, Fenton, Stoke-on-Trent, telephone 0782 273815.

WIN A FISCHERTECHNIK ROBOT KIT

Your Robot will be supporting the Fischertechnik robot building kit with a series of projects over the next few months – we think the kit offers an excellent introduction to the world of robotics.

We've two kits to give away in this month's **Your Robot** competition.

How to enter

In not more than 50 words describe an application for a robot based on the Fischertechnik robot kit.

There are two prizes to be awarded – one to an individual and the other to a group entry from a school or college.

Write your entry on a piece of paper together with your name and address – or that of your school in the case of a group entry – and send to Fischertechnik Competition, Your Robot, 155 Farringdon Road, London EC1R 3AD.

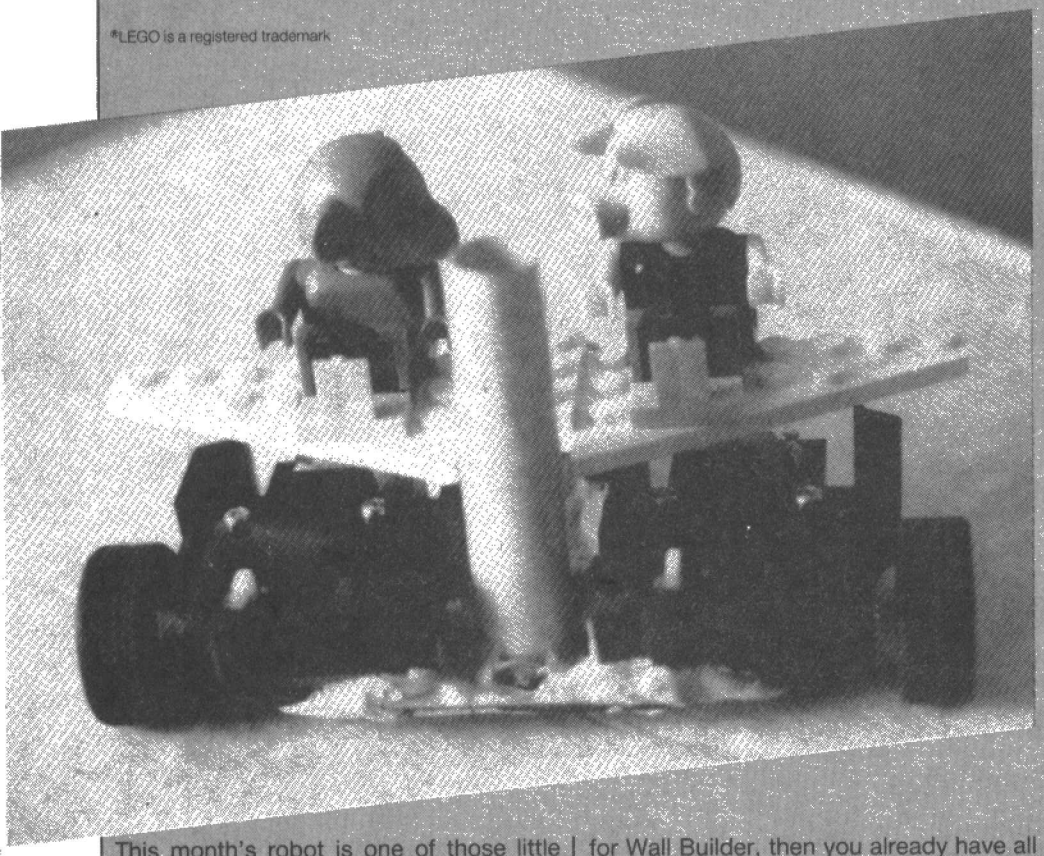
No EMAP employee may enter the competition and the editor's decision in all aspects will be final. Closing date for entries, October 31st.

Your Robot, 155 Farringdon Road, London EC1R 3AD. **Editor:** Gary Evans. **Deputy Editor:** William Owen. **Production Editor:** Liz Gregory. **Advertisement Manager:** Richard Jansz. **Advertisement Production:** Yvonne Moyser.

BUILDING A LOW-COST TURTLE

Richard Sargent shows how, for under £10, you can build a flexible turtle like device.

*LEGO is a registered trademark



This month's robot is one of those little trundling machines which draws geometric shapes on a very large piece of paper. As a species they're generally called turtles and are usually controlled by the high-level computer language, Logo. Our device is called Laurel and runs on Basic and, to keep things (more or less) simple, it can be run from a standard Centronics port following the guide lines laid down in the article which appeared in the August issue of **Your Robot.**

CONSTRUCTION

If you had all the LEGO® bits and pieces for Wall Builder, then you already have all that's needed for Laurel. Turtles are pretty standard devices in that they have two wheels in the same axis line, and a third "wheel" (it's usually a stud or castor) forming the apex of a triangle, see **Figure 1.** The bulk of the weight is inside the triangle because this arrangement affords good stability. The pen is midway between the two wheels and acts as a convenient pivot point when the device is turning. Each wheel has its own motor and can operate independently of the other. Turns are made either by stopping one wheel whilst allowing the other to continue, in which case any line drawn will be a curve with radius **R,** or by reversing the direction of one wheel whilst allowing the other to continue normally; on this occasion no line is drawn because the pen-tip is the centre point of the turning-circle.

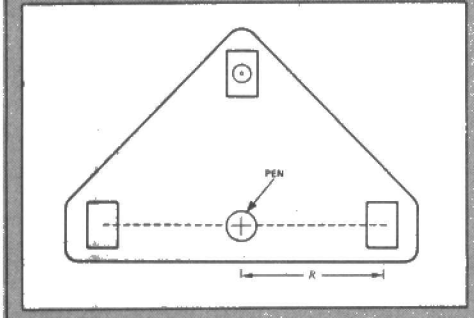


Figure 1. Plan view of a typical turtle.

The LEGO Turtle is clearly going to use the two 4.5V geared motors mounted side by side with just sufficient gap between them to hold a felt-tip marking pen. There need be no chassis as such – you merely use LEGO plates to join the motors together – and a ball-bearing glued into a square brick will function admirably as the third wheel. The axles and wheels are provided with the motor if you use the Kit 107.

CANNIBALISING CARS

Small motorised cars can be a useful source of materials, and the prototype Laurel was constructed from two "remote control" cars costing just £2 each. This sort of model car is always available under a variety of names and typically has a hand-held battery case complete with "gear lever". All that really matters is that it has a low-geared motor, sensible tyres, and a plastic body which can easily be cut to pieces with a small hacksaw and a Stanley-knife.

The low-cost of these motors certainly pays off if you're contemplating a 6-axis robot, but it is important that the ease of construction offered by LEGO is still available. Therefore, this month the construction details given dwell on the mechanical interface between cheap motors and LEGO, rather than an explicit description of the particular model, Laurel.

BODY BUILDING

There are two particular materials which are invaluable to the robot builder. One is a solid adhesive such as Blu-Tak, the putty-like medium which holds things firmly, but temporarily, while you decide if it all works. The other is Plastic Padding, a plastic compound in a tube which seems to stick to anything and which sets rock hard in two minutes. It has other virtues too: it doesn't shrink and it doesn't conduct electricity.

Laurel is a relatively easy construction since the original motors and gearing are

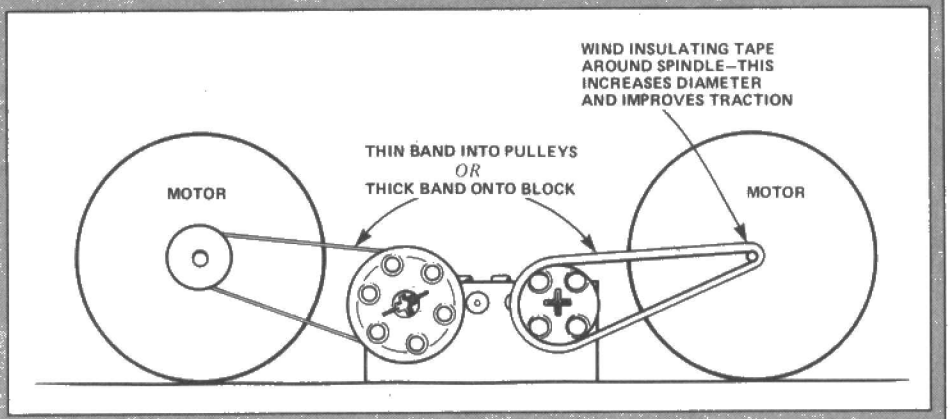


Figure 2 shows the use of rubber bands to aid the transmission of power.

not modified. The bodies of the two cars are cut away until the smallest size chassis remains. Then a plastic plate is used to tie the two chassis together. I used the black plastic from a cassette case. It is usually ready-scored in a graph-grid pattern and is easy to mark out and cut to size. This plate will need a hole cut in it and will eventually hold the pen-tube. My method of producing large diameter holes in plastic is to melt the plastic away with the tip of a soldering iron but whatever your choice of method – burning, drilling or cutting – take your time and exercise extreme caution. The plate should be held in a vice or clamped with a bulldog clip to a scrap piece of wood. Fingers and eyes are vulnerable during this operation so take care.

The pen slides in its own tube and the force of gravity is sufficient to keep it pressing down on the paper. A plastic roller-ball pen was used in the prototype and you can choose either a fine or a thick point with the pen's diameter being the same for either type. The tube is made from medium weight card wound around the pen and paper; glue was used between the layers of card.

The whole construction is finished by winding on adhesive tape as the last few layers. The tube is push-fitted into the hole in the plastic plate and a generous seal of Plastic Padding keeps the tube rigid and upright. The prototype Laurel has for its third wheel a piece of LEGO, glued and Plastic-Padded to the motor chassis. It is, of course, worth while adding the LEGO even if you have to cut some of it to make it fit, since then you have the LEGO studs onto which other useful (or decorative) pieces of equipment can be fitted.

AWKWARD MOTORS

If you are prepared to accept a larger chassis size, then you can press almost any type of motor into service and use TECHNICAL LEGO to reduce the rpm to sensible proportions. **Figure 2** shows how elastic bands might be used to transmit power

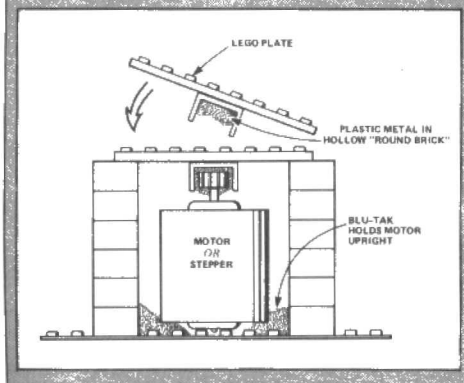


Figure 3. A "round brick" will pass power directly to a LEGO axle.

from the motor to the LEGO system. Rubber bands slip, of course, and so you might try the direct approach of bonding LEGO to the motor shaft. It is important to position the LEGO piece accurately, and to this end a jig is constructed which will more or less do the job for you. A hollowed-out LEGO "round brick" is probably the best piece to fit on the shaft, since it will be able to pass the power directly to a LEGO axle. **Figure 3** shows how this is done.

If your motor requires an unduly high DC voltage or (horror!) an AC voltage then you will need to replace the transistor-switching circuit shown last month with an equivalent circuit using relays. This is shown in **Figure 4.** The disadvantage of using relays is that they will not pass the low-frequency pulses which are produced by software to turn the motors slowly. Slow speed will therefore have to be created by suitable gearing.

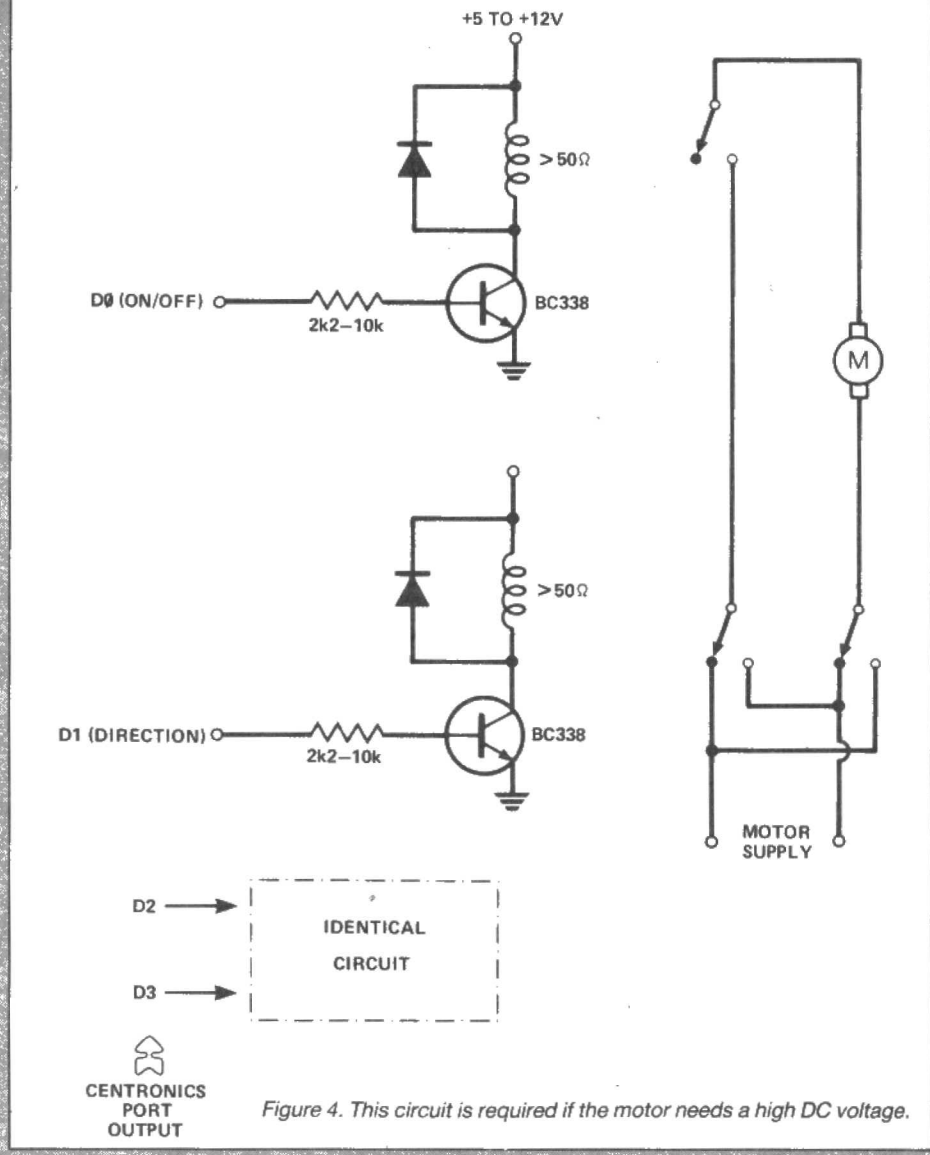


Figure 4. This circuit is required if the motor needs a high DC voltage.

"This turtle runs on Basic and will operate from a standard Centronics port".

DRIVING LESSONS

Laurel is essentially driven in the same manner as Wall Builder, and a short Basic program will allow you to control its movements from the keyboard, using LPRINT commands. The control byte sent to LPRINT as CHR$(x) is calculated as follows:

Shut down	00000000	0
Left motor forward	00000001	1
Left motor reverse	00000011	3
Right motor forward	00000100	4
Right motor reverse	00001100	12

If you assign keys to these codes, you can have a test drive:

```
Q = Quit drive

A=left forward        S=right forward
Z=left reverse        X=right reverse
Space bar=stop

100  LET V$=INKEY$
110  IF V$="Q" THEN STOP
120  IF V$="A" THEN LPRINT CHR$(1)
130  IF V$="Z" THEN LPRINT CHR$(3)
140  IF V$="S" THEN LPRINT CHR$(4)
150  IF V$="X" THEN LPRINT
     CHR$(12)
150  IF V$=" " THEN LPRINT CHR$(0)
160  GOTO 100
```

Use 100 INPUT V$ if your Basic lacks an INKEY$

You will quickly find that this program, although fun, leaves a lot to be desired. It's a pity that Basic can't detect two simultaneous key presses so that both motors

could be started at once. Extra lines such as:

```
122  IF V$="Q" THEN LPRINT CHR$(5)
124  IF V$="C" THEN LPRINT
     CHR$(15)
```

will provide this facility, but to do the same for turning and pivoting simply means having to remember more and more keypresses, and you will end up looking at the keyboard 90% of the time. To overcome these and other problems, turtles have their own command language which enables them to be driven efficiently, to perform automatic sequences of movement, and to remember entire "journeys".

MOVING GRAPHICALLY

A Command Language can be written in Basic and tested on the computer screen before letting loose Laurel to blunder its way across that square yard of pristine-white sugar-paper. If you have a Spectrum or a Dragon you might like to dig around for *E&CM* May 1983 and read Mike James' article "The Computer Brain", where an idea known as "chain coding" is discussed at some length. Chain coding is the basis of languages like Logo and is also the concept behind the Dragon's graphics commands. When applied to crawling robots it gives a list of movement instructions which are obeyed in strict order to build up shapes and patterns in the way that humans would draw them, or pace them out on the ground. Thus a Logo listing might read more like instructions for finding buried treasure than the more old-fashioned Basics which work on a great number of meaningless screen co-ordinates.

"Any extension to Laurel's command language is very simple".

LOGO DRAWS A SQUARE

```
START AT 20,20
REPEAT 4 TIMES:
   TURTLE FORWARD 10 UNITS
   ROTATE TURTLE 90 DEGREES
   END REPEAT
END PROGRAM
```

The above is a program which could be considered elegant. In reality the listing would be a shorthand version so that it was economical in its use of computer memory. If the starting co-ordinates were omitted, the square would be drawn from the turtle's present position.

Logo has recently been implemented on a number of home micros, including the Sinclair Spectrum and the Amstrad CPC464. We will have to be content with a small interpreter, written in Basic, which uses chain code to drive the Laurel Turtle. The code is written in MTX Basic which is a pretty standard Basic, and uses RIGHT$ and LEFT$ to step through the turtle-movement instructions which are fed by the user into B$. B$ can be as long as your computer will allow and is a string made up of <letter,number(s)> sequence, entered without spaces. Letters should be upper case, and numbers should be integers. At least one digit should separate each letter: there is no error-checking in this simple program, so you must enter the instruction codes carefully into B$.

Legal letters are F,B,R,L and S for Forwards, Backwards, Right turn, Left turn and Stop – 5 commands in all. Each command has a subroutine, numbered 0-4, assigned to it, and the "command table" of legal letters and subroutines is contained in the DATA statement at line 210. The first item, 5, is the number of commands. Then comes each command letter, followed by its subroutine number. Line 200 directs the program flow to the correct subroutine. Subroutine 0 is at line 300, subroutine 2 is at line 400 and so on. Any extension to Laurel's command language is very simple: the new letter and subroutine number are tacked onto the end of the DATA statement, and the appropriate subroutine is allocated (line 200) and then of course, written.

The numbers which are given after the commands F and B represent units of time during which Laurel's motors are ON. A single unit of time has a length proportional to the value of MD (line 42). Commands R, L and S do not use numbers, but require some dummy value to be typed in. The values RD and LD in line 40 should be adjusted until it represents a time period which causes Laurel to turn through ninety degrees. Obviously the values RD, LD and MD will vary depending on the model of computer you are using, and their adjustment is very much a case of trial-and-error.

Laurel is too small to have sensors fitted to it, and it's a nuisance not being able to raise its pen to stop its tracing activities: next month we'll return to LEGO with a vengence and start building a 4-motor floor buggy which **can.**

LISTING 1. A small turtle interpreter.

```
10 REM SIMPLE LAUREL RECOGNISES F - FORWARD   B - BACKWARD
20 REM                          R - RIGHT 90' L - LEFT 90'
30 REM                          S - STOP
40 LET RD=20: LET LD=20: REM RD&LD ARE THE TURN DELAYS
42 LET MD=20: REM MD IS THE ONE-UNIT-OF-MOVE DELAY
60 CLS : READ COM: DIM A(COM)
70 FOR N=1 TO COM
75 READ A$(N): READ A(N): NEXT N
80 INPUT B$: IF B$="0" THEN STOP
90 IF B$="" THEN GOTO 80
100 LET N=1: LET X=LEN(B$)
110 IF B$(1)=A$(N) THEN GOTO 140
120 LET N=N+1: IF N>COM THEN GOTO 80
130 GOTO 110
140 LET B$=RIGHT$(B$,X-1): LET K=1
150 IF RIGHT$(B$,X-K)<"0" OR RIGHT$(B$,X-K)>"9" THEN GOTO 180
160 LET K=K+1: GOTO 150
180 LET K=K-1: IF K=LEN(B$) THEN LET V=VAL(B$): GOTO 200
181 IF K=LEN(B$) THEN LET V=VAL(B$): GOTO 200
190 LET V=VAL(LEFT$(B$,X-K-1)): LET B$=RIGHT$(B$,X-K-1)
200 ON A(N) GOTO 300,400,500,600,700
210 DATA  5,F,0,B,1,R,2,L,3,S,4
219 REM MOVE TIME PERIOD
220 FOR Z=1 TO V
222 FOR W=1 TO MD: NEXT W
224 NEXT Z
226 RETURN
279 REM TURN TO RIGHT TIME PERIOD
280 FOR W=1 TO RD: NEXT W
282 RETURN
289 REM TURN TO LEFT TIME PERIOD
290 FOR W=1 TO LD: NEXT W
292 RETURN
299 REM FORWARDS
300 LPRINT CHR$(5): GOSUB 220: LPRINT CHR$(0): GOTO 90
399 REM BACKWARDS
400 LPRINT CHR$(15): GOSUB 220: LPRINT CHR$(0): GOTO 90
499 REM RIGHT 90'
500 LPRINT CHR$(4): GOSUB 280: LPRINT CHR$(0): GOTO 90
600 LPRINT CHR$(1): GOSUB 290: LPRINT CHR$(0): GOTO 90
699 REM STOP
700 LPRINT CHR$(0): GOTO 90
599 REM LEFT 90'
```

FISCHERTECHNICK ROBOTS

Richard Sargent has built one of the devices from the range of robots that can be constructed from the new Fischertechick robot builder's kit.

Few people, I suppose, have had occasion to use Fischertechnick as a building medium unless they are involved in the teaching of Science, or have made up or otherwise modified ECONOMATIC's BBC-Buggy, which is made largely of Fischertechnik parts. Last month the new Robot package from the German firm was previewed, and since then quite a few of us have been brushing up on our German and generally getting the feel of the kit by attempting to build one or more of the robot designs presented.

THE KIT

The kits on sale to the public will, of course, have their instructions written in clear, unambiguous English. To refresh your memories, here is the list of *explained* models which can be built up from the parts in the standard kit. There are naturally the proverbial 101 other designs which will occur to lateral thinkers (or those of you with time on your hands), but even budding Einsteins have to start somewhere!

A "Tower of Hanoi"-playing Robot.
A Solar tracking device.
A pen plotter.
A sorting machine.
A graphics tablet.
A 2-axis Robot arm with magnetic gripper.

The kit is comprised of the plastic components, including two motors and two reduction gear-blocks, eight micro switches, two potentiometers, four lights, a great many little plugs (they go into the motors and the switches), some ribbon cable, and, thoughtfully, a small screwdriver for the plugs! Cynics will note that although a felt-tip pen is supplied for "Pen-Plotter", a solar cell is *not* provided for "Solar-Tracking Device"!

THE BITS

The Fischertechnick building medium is easy to use and creates a very rigid final model. Inevitably comparisons with LEGO will be made, but the method of linkage between the Fischertechnick pieces is quite different, and it remains to be seen which of the two systems will be preferred for different types of model robot. The Fischertechnick Robot Builder kit lacks wheels and axles, and the LEGO **ROBOTS** currently being described in **YOUR ROBOT** could therefore not be built using Fischertechnick.

The Robot Builder Kit is particularly well suited to the construction of open, vertical structures. The basic Fischertechnick piece is the channelled bar, the cross-section of which is always square (see **Figure 1**). At its smallest size the bar is a 1.5cm cube and at its longest it is a beam 18cm in length. **Figure 2** shows the method of linkage between one bar and another. Unlike LEGO with its studs, Fischertechnick has no "top surface" and no "undersurface". You can use it efficiently whichever way round it is angled. The bar-channels are cunningly designed to accept axles and there is no necessity for separate "technical" bars or beams. The axles are made of metal and they run very smoothly in the channels. They can also be used as link-pieces to help join the bars together and this is an important factor where the strength of the final model is an overriding consideration.

AND (SPECIAL) PIECES

The Fischertechnick motor is a small but powerful beast, and runs happily on a 5V DC supply. It can be integrated into your model easily and its small size (60x20x15mm including reduction gears)

SOME THOUGHTS ON INTERFACING

There are a number of different ways in which the two motors of the Fischertechnik robot building kit can be interfaced to a microcomputer system. The kit began life in Germany as a complement to a Z80 based micro trainer unique to the German market. The native instruction manual provides full details of the interface electronics and software necessary to drive the robots from this computer. Although this information is of little direct use it does provide some ideas as to how interfacing may be accomplished.

The method chosen is a fairly software intensive scheme and the software shown is all written in machine code. The motors are driven from a fairly standard looking Z80 output port, the control lines driving a transistor which in turn controls a relay that switches power to the motor. It is in the area of providing feedback information from the robot's potentiometers that the circuit and software get a little more complex.

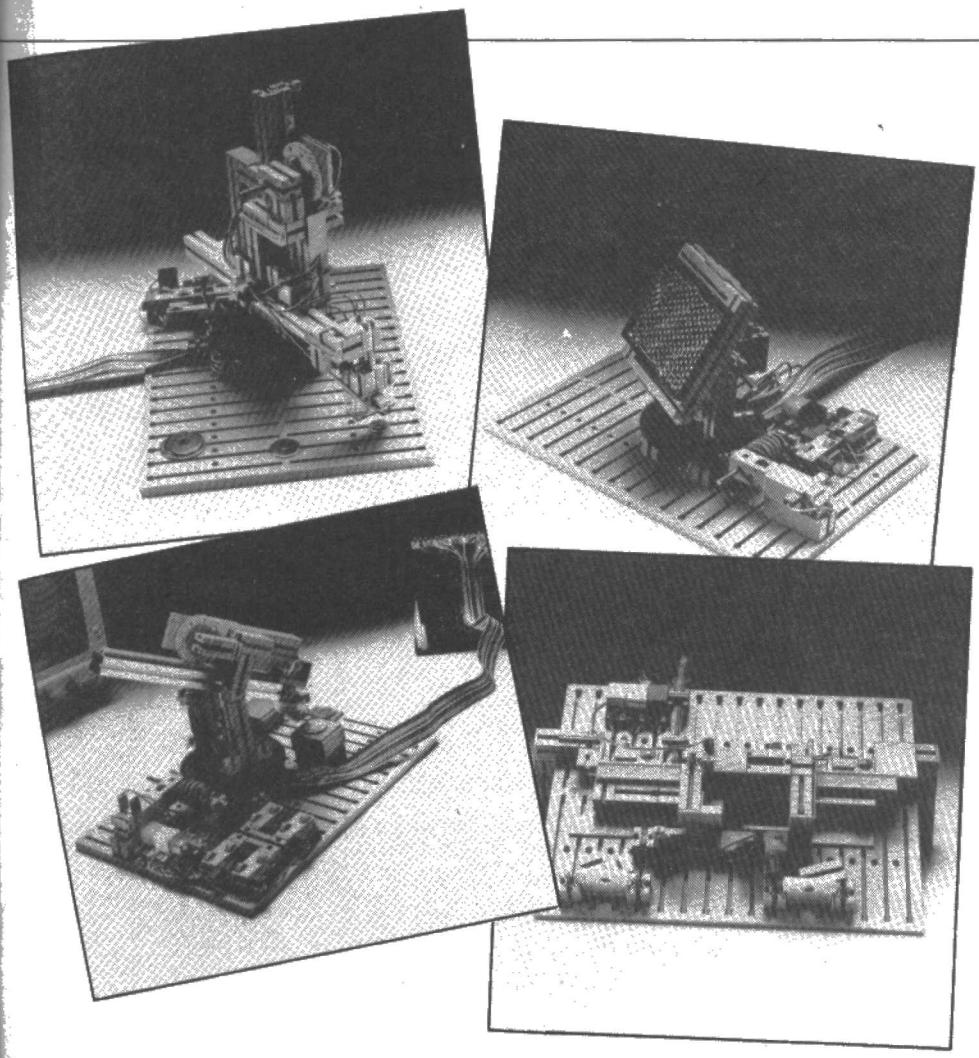
The method adopted is to use the pots to control the output of a voltage to frequency converter. It is then the job of the software to determine from the input frequency the position of the axes associated with the particular pot.

This method keeps hardware to the minimum, although a further saving could be made by replacing the transistor/relay combination with a power driver, but this has a number of disadvantages if a general purpose interface is required. While the hardware would probably be capable of being tailored to different micros with minor if any modification, this is not the case with the software, which would require quite extensive modifications if it were to be used on any other computer.

When designing an interface for a computer that is already equipped with an A/D converter, as in the case of the BBC and Dragon computers, the problems are lessened. The Micro Robotic Systems interface uses the BBC's user port output lines to control the two motors via a series of opto-isolators. The signals from the two pots are simply fed to the inputs of the computer's A/D converters and the software does the rest. By virtue of the fact that the routines for dealing with the A/D conversion provided in the computer's firmware, the software overhead is much reduced.

The disadvantage with this approach is that it is machine specific. While many experimenters in schools and colleges will have access to a BBC computer, by no means all will, and the Micro Robotic's interface and software will be of little use to owners of other computers.

A general purpose interface could be constructed along the lines of the design



Michael Graham puts forward a few ideas of hooking the robots up to a computer.

produced by Richard Sargent for his series on LEGO robots currently appearing in **Your Robot.** This interface is designed to connect to the Centronics printer interface of the controlling computer. The Centronics interface is to be found on the majority of home micros and on those computers that do not feature it; the most notable example being the Spectrum; it is a fairly simple matter to implement one with little more than a Z80 PIA. The disadvantage with designing an interface around the Centronics interface is that there is no route by which the feedback signals from the pots may be fed to the controlling computer.

This difficulty can be overcome by borrowing the ideas incorporated in Powertran's interfzce for their micro grasp robot arm. This used an interface that featured much more hardware than those so far described.

The basis of this interface is that the computer merely outputs an eight bit binary value that corresponds to the required position of the controlled axis and the hardware is then responsible for carrying out the movement with no intervention on behalf of the computer. Basic operation is as follows: The binary number output by the computer is latched and then fed to an A/D converter. The signal from this converter is then compared with a voltage derived from one of the feedback pots in a fairly standard bridge network. The bridge will attempt to reach a balance by driving the motor in the appropriate direction. Careful attention to the damping network of the bridge can provide a fast and accurate response for the total system. An additional refinement would be to feed back a 'busy' signal to the computer corresponding to an out of balance condition in the bridge. Such lines are provided as part of the Centronics interface and this facility would mean that if necessary the computer could be used to read the position of a pot in a device such as the digitiser by outputting all possible positional values until the busy line became inactive indication that the value output correspond to the position of the pot.

The attraction of the last scheme described is that a truly general purpose interface may readily be constructed and that a minimum of software is necessary to drive the robot system, the computer merely being responsible for poking a number to the printer port and possibly monitoring the state of the busy line.

Your Robot will be looking at the various interfacing ideas presented here in the next few months. They will have many applications behond the Fischertechnick robot kits and will be of value to anyone experimenting with computer control systems.

makes it possible to incorporate it into the structure of, for example, a robot arm. It is particularly efficient at driving a rack-and-pinion arrangement, or, by means of a worm-gear, driving a large turn-table unit. But perhaps the most useful pieces of all in the kit are the two potentiometers and their

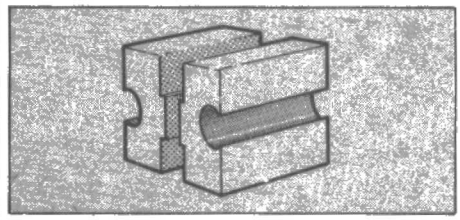


Figure 1. Bar piece.

holders. These provide the all-important feedback signal to the computer which can be used to calculate the robot's actual position. This allows for great accuracy when the model is running – the software knows exactly where the mechanical bits and pieces are at any given moment, and cumulative errors caused by friction, excessive loading of joints, or even accidental sabotage by an inquisitive finger or two, can be detected and corrected.

At least one of the potentiometers has full 360 degree travel, and gives a valid analogue voltage from all but about 30 of those degrees. However, your interface might not have the facility for Analogue to Digital conversion, in which case you would need to fall back on the micro-

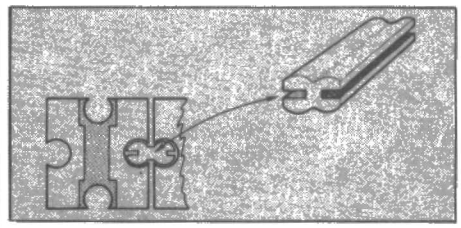


Figure 2. Link method.

switches supplied and use them as limit switches to detect the end-of-travel situations. Also in the kit is a Fischertechnick electro-magnet. The coil resistance is 30 ohm and 5volts is sufficient to drive it.

WHAT, NO MICRO?

As is the case with LEGO, you can have many hours of fun running your Fischertechnik model directly from a low-voltage DC power supply or from three of four 1.5V batteries connected in series. Alternatively you can drive it successfully from the same interface that is used to run the LEGO robots. If you have a BBC B machine you will be able to read the robot's potentiometers and evolve some simple software for making your model do some clever things. There is no doubt that the existence and availability of this type of robotics kit in Britain will enable people to take an active interest in computer-control experiments, and when the purpose-built interface and demonstration listings from Micro Robotic Systems Ltd are available we'll put some of Fischertechnik's little Robots through their paces. Perhaps a small workshop of Fischertechnik androids could help finish the LEGO ROBOTICS series . . .

INTERFACE

Peter Luke puts pilot one's BBC control interface through its paces. The robust design has been produced to meet the requirements of the educational market.

While Pilot One may be accused of a certain lack of imagination when it came to the naming of their new product, the chosen handle does have the advantage of simplicity. The name chosen was Interface and that term exactly describes the product. The unit connects to the user port of the BBC micro and provides four input and four output lines.

Interface is housed in a two tone Vero box that will be a familiar sight to anyone who has had cause to find a home for one of their own DIY electronic projects. Interface has its own internal power supply and judging by the weight of the box, the transformer chosen for the design is quite generously rated. This reflects the overall design philosophy behind Interface; it has been built to last and is as far as possible both child and idiot proof. Connection to the outside world is by way of a series of banana sockets mounted on either side of the unit. Each of the four input and four output lines has three sockets associated with it. In the case of the input side these are the three connections to a changeover relay.

The output lines also have connections to the +5V power supply brought out to the same side of the box. The three input connections correspond to the input to a schmitt circuit in addition to the ground and +5V power lines.

All of the lines are buffered and whatever accidents may happen during the course of driving equipment via the interface it is unlikely that any harm will come to the BBC micro.

The top of the box has a series of mimic LEDs that give an immediate indication of the state of both the input and output lines.

MANUAL DRIVE

The manual that accompanies Interface provides a clear explanation of the operation of the BBC micro's user port and the way in which Interface works in conjunction with the eight I/O lines of this port including a description of the Data Direction Register. The manual goes on to describe some typical circuits that can be used to condition both input and output signals. In terms of inputs these include circuits to switch a buzzer on and off, a simple on/off motor driver and a circuit that allows a motor to be driven both forwards and backwards. Input circuits include those for both touch sensors and light sensors, the latter based around a MEL11 photodiode.

"Interface should prove robust enough for most environments".

Interface is supplied with a suite of software. This includes a collection of PROCedures incorporating the POKEs necessary to control the relays within easy to understand blocks of BASIC programs.

The collection of software also includes a program called 'INTCAR' which allows the interface to run a two motor model car. The software assumes that the left motor is connected to lines PB0/PB1 and the right motor to PB2/3. The program allows the user to design a course on the screen and then to store this on tape or disk. The stored course can then be relayed either on screen or, through the interface, using a model car (the buggy described in this issue of **Your Robot** would be ideal).

Pilot One can also supply a crane and model Jeep suitable for use with Interface (these are pictured on the front cover). The crane in particular provides an impressive illustration of the way in which computers can control the operation of a variety of equipment and in the way in which the physical action of equipment is related to the digital information output by the computer.

The Pilot One Interface is built to a high standard and should prove robust enough even for the most hostile of environments ie school classrooms. The software supplied allows the interface to be programmed without recourse to machine code programming although it makes sufficient information available for those that may wish to control it in this way.

Pilot One
Victoria House
46 St. Augustines Road
Bedford
MK40 2ND

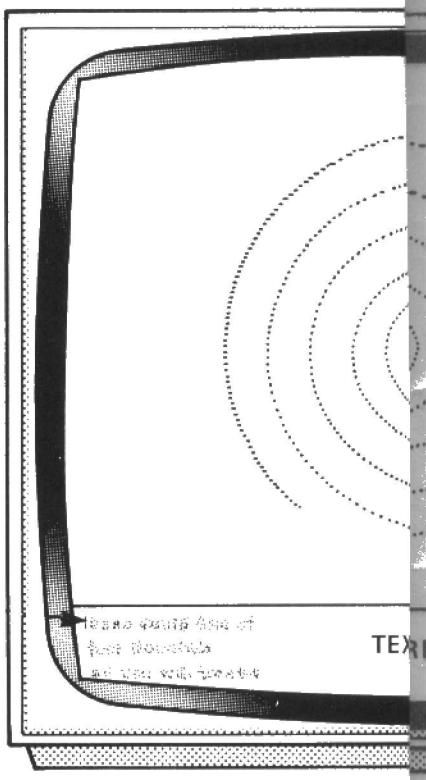
Interface specifications

Internal power supply	1 Amp-short circuit protected
Output lines	Single pole c/o relays rated at 0.5A, 50V
Input lines	Opto-isolated (isolation voltage 250V min). Drive current 10mA min.

LED indication on all I/O lines.
Load on BBC micro – 12mA with all lines at +5V.

COMMODORE 64 SPLIT

Ken McMahon uses interrupts to split the Commodore 64's screen display into two windows – one in text mode, the other in multi-colour bit mapped mode.



One of the more useful features of the Commodore 64's screen format is the ability to use any one of three available screen modes depending on whatever particular application you have in mind. These are standard (or extended) text mode, bit map mode, and multicolour bit map mode.

All very useful. But supposing you want to use more than one mode on the screen at the same time? For example, you might want to display hi-res graphics information on one part of the screen with accompanying text on another. This can be done with the aid of a short machine code routine which uses interrupts to change the configuration of certain registers in the VIC-II chip as and when required.

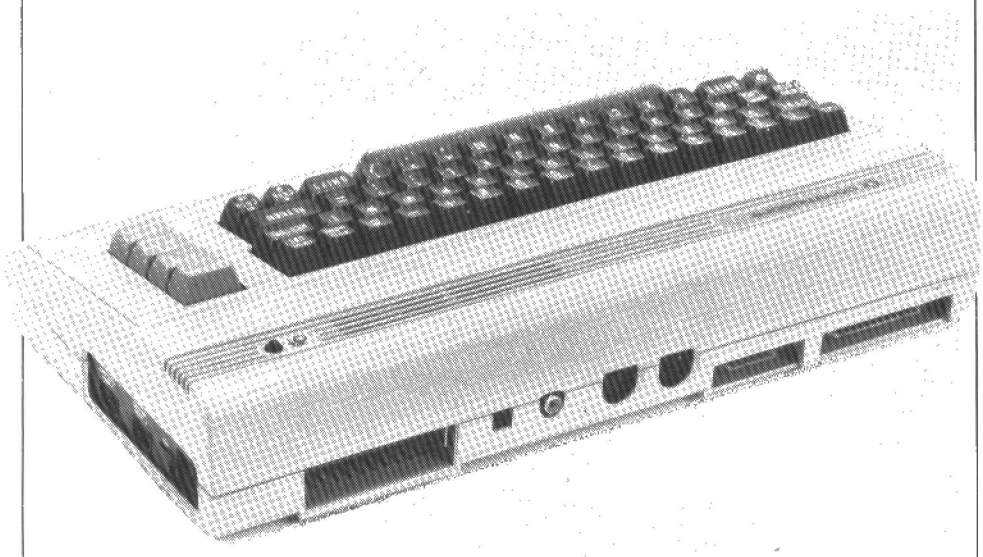
In the main the registers we will be working with are:

1 The raster compare register at location 53266 (HEX D012).
2 The VIC control register at location 53265 (HEX D011).
3 The VIC interrupt flag register at location 53273 (HEX D019).
4 The interrupt mask register at location 53274 (HEX D01A).

The raster compare register tells the computer precisely where the screen is being drawn at any given moment. By writing a value to this register we can cause an interrupt (IRQ) to occur at a point when the position of the screen is equal to the value that has been written in.

The control register determines the screen mode depending on the bit configuration within it. The particular bit we are interested in is bit 5 which enables bit map mode. The last two registers are both concerned with interrupts. The interrupt flag register sets certain bits when interrupts have occurred. The bit set depends on what triggered the interrupt. This time we are interested in bit 0 which is the raster compare IRQ flag. Correspondingly, bit 0 of location 53274 is the raster compare IRQ mask bit; this must be set or the computer will ignore the raster interrupt when it occurs.

If you do not have a machine code editor/assembler you will need to type in the whole of the BASIC listing program in **Listing 1.** (Remember to save the program before running it: the checksum is not infallible!) Otherwise the assembly code can be typed in direct, although lines 200-340 of the basic program will still need entering in as these calculate the data for the graph and format the text output.



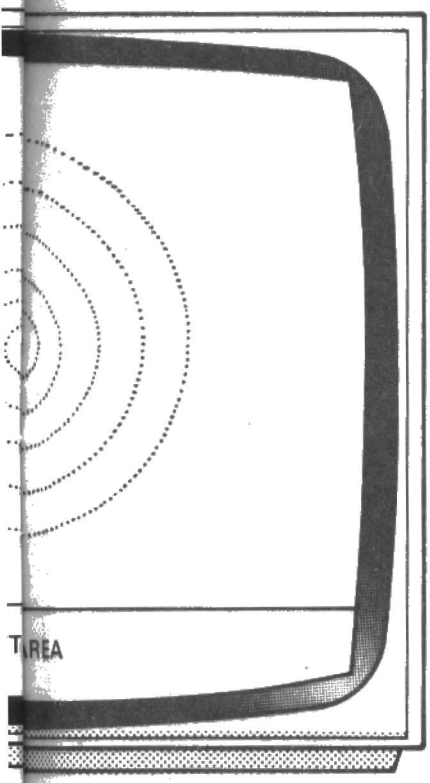
When run the program will clear the screen down to the bottom three lines and start to draw the hi-res image – a spiral. The x and y co-ordinates of each point are printed at the bottom of the screen as they are plotted.

```
100  C000 LDA £;$00
110  C002 STA Z;$FB
120  C004 LDX £;$20
130  C006 STA Z;$FC
```

The first task is to assign an area of memory (8K) for the hi-res screen. The area to be used starts at location 8192 ($2000). This address has been deposited in page zero as it will need clearing later by filling it with zeros. The most economical way of doing this is to use indexed indirect addressing. This method also has the advantage that should the screen need to be moved elsewhere, all that has to be changed is the address pointer at location $FB. Register A has been used to load $FB and register X to load $FC; the reason for this will become apparent later.

```
140  C008 LDY £;$00
150  C008 STA(I).Y;$FB
160  C00C INY;
170  C00D DNC,$FD
```

Now the screen area can be cleared. Line 150 stores the contents of register A, which has already been loaded with 0, at the memory address pointed to by the turn page location $FB plus a displacement, Y. On each pass of the loop, Y is incremented to point to the next location. After 256 pas-



ses incrementing Y will cause it to equal 0, the condition in line 170 will no longer hold and the branch will not occur.

```
180   C00F INC Z;$FC
190   C011 DEX;
200   C012 BNE;$F6
```

This is all very well, but only 256 bytes have been cleared and we need to do over 8000. Line 180 increments the high byte of the address pointer – an effective increment of 256. We are now ready to clear the next 256 byte block but before this can be done the block counter must be decremented. You will recall that I mentioned earlier there was a reason for using register X to load location $FC with the high byte of our address pointer. In order to fill 8K of memory with zeros 32 * 256 byte blocks must be filled. By coincidence register X now contains $20 (=32 decimal), so all we have to do is decrement it each time a 256 byte block of memory is cleared. When this has occured 32 times X will decrement to 0, the condition in line 200 will no longer hold and we can go on.

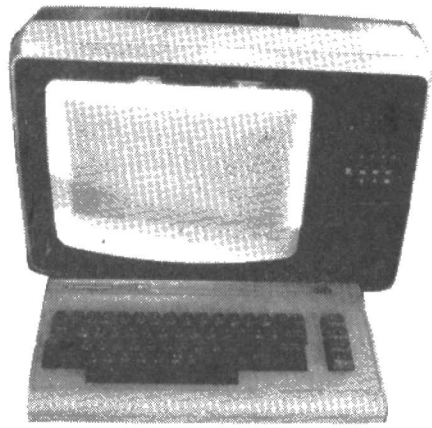
```
210   C014 LDX £;$04
220   C016 STX Z;$FC
230   C018 LDA £;$03
240   C01A LDY £;$00
250   C01C STA(I(.Y;$FB
260   C01E INY;
270   C01F BNE;$FB
280   C021 INC Z;$FC
290   C023 DEX;
300 C024 BNE;$F6
```

Having cleared the screen area the foreground and background colours for the hi-res image must be set.

In bit map mode these are determined for each eight by eight bit square by the contents of locations 1024 – 2023 ($0400-07E7), the place in memory usually occupied by the screen. The high four bits of each location control foregound colour; the low four, background colour. The same method used to clear the hi-res screen area is adopted. Our address pointer in page zero is loaded with address 1024 ($0400) via register X which is also used as a block counter (4*256=1024). Each location from 1024 to 2023 is loaded with the value 03 which sets the foreground and background colours to black and cyan respectively.

```
310   C026 SEI;
320   C027 LDA £;$3B
330   C029 STA;$0314
340   C02C LDA £;$C0
350   C02E STA;$0315
360   C031 CLI;
```

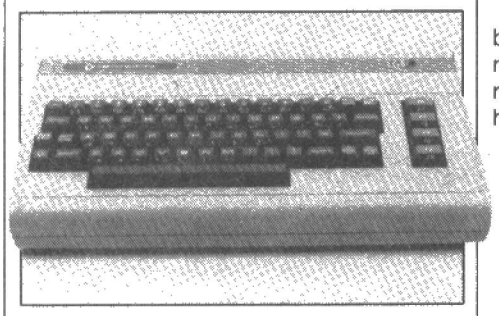
Now we can get down to the real work. Locations $314 and $315 contain the



LISTING 1. BASIC listing program.

```
10 BASE=49152:FOR N=BASE TO BASE+123
20 READ A:POKE N,A:C=C+A
30 NEXT
40 IF C<>16379THEN500
50 DATA169,0,133,251,162,32,134,252,160,0,145,251,200,208,251,230,252,202,208
60 DATA246,162,4,134,252,169,3,160,0,145,251,200,208,251,230,252,202,208,246,120
70 DATA169,59,141,20,3,169,192,141,21,3,88,173,26,208,9,1,141,26,208,96,173,25
80 DATA208,41,1,240,55,141,25,208,173,18,208,201,208,144,24,169,1,141,18,208,173
90 DATA24,208,41,247,141,24,208,173,17,208,41,223,141,17,208,76,49,234,169,224
100 DATA141,18,208,173,24,208,9,8,141,24,208,173,17,208,9,32,141,17,208
110 DATA76,49,234
200 PRINT"[CLR/HOME]"
210 SYS49152
215 POKE53265,PEEK(53265)AND127
220 PRINT"[CURSOR DOWN] * 25"
230 FORK=0TO54.7STEP0.1
240 X=INT(150+3*K*SIN(K)):Y=INT(100+2*K*COS(K)):REM SPIRAL FORMULA
245 REM BIT MAP DATA CALCULATION
250 CHAR=INT(X/8)
260 ROW=INT(Y/8)
270 LINE=YAND7
280 BYTE=8192+ROW*320+8*CHAR+LINE
290 BIT=7-(XAND7)
300 POKEBYTE,PEEK(BYTE)OR(2^BIT)
310 PRINT"VALUE X    ";X,"  VALUE Y";Y;"[CURSOR LEFT][SPACE]"
320 PRINT"[CURSOR UP]
330 NEXT
340 GOTO340
500 PRINT"DATA ERROR:CHECK CODE"
510 STOP
READY.
```

Commodore 64's hardware interrupt vector. Whenever an interrupt occurs the processor branches to the two byte address contained at those locations. The vector needs to be altered so that it points to our program which begins at $C03B. Before doing this it is necessary to disable the interrupts using the SAI instruction (which somewhat ambiguously stands for SEt Interrupts). The reason for this is to prevent an interrupt occuring when we are half way through the process of changing the vector; this would result in havoc.



Having redirected the vector the interrupts can be enabled using the instruction CLI (Clear Interrupts).

```
370  C032 LDA;$D01A
380  C035 ORA £;$01
390  C037 STA;$D01A
400  C03A RTS;
```

One last thing. We must set bit 0 of the interrupt mask register at location $D01A otherwise the raster interrupt, when it occurs, will be ignored. Exit the program in the usual way with an RTS instruction.

```
410  C03B LDA;$D019
420  C03E AND £;$01
430  C040 C040 BEQ;$37
```

So far we have cleared an 8K area of memory for the hi-res screen, set the colours, and altered the hardware IRQ vector to point to a program at $C03B. Well, this is it. Every time an interrupt occurs the computer will come to $C03B, and it is this part of the program which will tell it what to put on the screen and where. The first thing to be determined when an interrupt has occurred is what triggered it, or, more precisely, was it the raster compare IRQ? If it was, the raster compare IRQ flag will be set and it can be tested with an AND instruction in line 420.

If it wasn't the raster which triggered the interrupt then it must have been something else and we branch out of the program to location $EA31 which is the IRQ vector to the usual interrupt handling routines.

```
440  C042 STA,$D019
```

Now we know that this is the real thing, and we must clear the raster IRQ flag. This is achieved by writing to it.

```
450  C045 LDA;$D012
460  C048 CMP £;$D0
470  C04A BCC;$18
```

By reading the raster compare register we can discover where the screen was being drawn when the interrupt occurred. If the value in the raster compare register is greater than $D0 then we will be in the bottom three lines of screen – where we want to put our text. The carry flag will be set and the branch in line 470 will not occur. Conversely if the value in the raster compare register is less than $D0 we will be in the top part of the screen and the program will branch to $C064 which configures the registers for bit map mode.

```
480  C04C LDA £;$01
490  C04E STA;$D012
```

Having determined that we are at the bottom of the screen the raster compare register must be set so that the next interrupt occurs when the top of the screen is hit – at $01.

```
500  C051 LDA;D018
510  C054 AND £;$F7
520  C056 STA;$D018
530  C059 LDA;$D011
540  C05C AND £;$DF
550  C05E STA;$D011
560  C061 JMP;$EA31
```

This part of the program simply sets the Commodore 64 to standard character mode. Lines 500-530 tell the VICII chip that the screen is at the usual place (1024-2023). Lines 540-560 turn off bit 5 of the VIC control register which, when set, enables bit map mode.

This is in effect the same thing as 'setting' standard character mode. Line 560 sends the processor to location $EA31 which is where it would have gone in the first place had it not been interrupted.

```
570  C064 LDA £;$E0
500  C066 STA;$D012
570  C067 LDA;$D018
600  C06C ORA £;$08
610  C06E STA;$D018
620  C071 LDA;$D011
630  C074 ORA £;$20
640  C076 STA;$D011
650  C079 JMP;$EA31
```

This part of the program deals with top screen work and does almost exactly the opposite of the preceding eight lines.

Now that we are at the top of the screen the raster compare register must be set so the next interrupt occurs at the bottom – $E0. The registers must the be configured for bit map mode. Lines 590-610 tell the VICII chip that the hi-res screen area is at location 8192 ($2000). Lines 620-640 set bit 5 of the VIC control register; ie set bit map mode. Once again we exit via the normal IRQ vector at $EA31.

LISTING 2. The machine code listing.

```
100  C000  LDA #;$00        A9 00
110  C002  STA Z;$FB        85 FB
120  C004  LDX #;$20        A2 20
130  C006  STA Z;$FC        86 FC
140  C008  LDY #;$00        A0 00
150  C00A  STA(I).Y;$FB     91 FB
160  C00C  INY;             C8
170  C00D  BNE;$FB          D0 FB
180  C00F  INC Z;$FC        E6 FC
190  C011  DEX;             CA
200  C012  BNE;$F6          D0 F6
210  C014  LDX #;$04        A2 04
220  C016  STX Z;$FC        86 FC
230  C018  LDA #;$03        A9 03
240  C01A  LDY #;$00        A0 00
250  C01C  STA(I).Y;$FB     91 FB
260  C01E  INY;             C8
270  C01F  BNE;$FB          DD FB
280  C021  INC Z;$FC        E6 FC
290  C023  DEX;             CA
300  C024  BNE;$F6          D0 F6
310  C026  SEI;             78
320  C027  LDA #;$3B        A9 3B
330  C029  STA;$0314        8D 14 03
340  C02C  LDA #;$C0        A9 C0
350  C02E  STA;$0315        8D 15 03
360  C031  CLI;             58
370  C032  LDA;$D01A        AD 1A D0
380  C035  ORA #;$01        09 01
390  C037  STA;$D01A        8D 1A D0
400  C03A  RTS;             60
410  C03B  LDA;$D019        AD 19 D0
420  C03E  AND #;$01        29 01
430  C040  BEQ;$37          F0 37
440  C042  STA,$D019        8D 19 D0
450  C045  LDA;$D012        AD 12 D0
460  C048  CMP #;$D0        C9 D0
470  C04A  BCC;$18          90 18
480  C04C  LDA #;$01        A9 01
490  C04E  STA;$D012        8D 12 D0
500  C051  LDA;$D018        AD 18 D0
510  C054  AND #;$F7        29 F7
520  C056  STA;$D018        8D 18 D0
530  C059  LDA;$D011        AD 11 D0
540  C05C  AND #;$DF        29 DF
550  C05E  STA;$D011        8D 11 D0
560  C061  JMP;$EA31        4C 31 EA
570  C064  LDA #;$E0        A9 E0
580  C066  STA;$D012        8D 12 D0
590  C069  LDA;$D018        AD 18 D0
600  C06C  ORA #;$08        09 08
610  C06E  STA;$D018        8D 18 D0
620  C071  LDA;$D011        AD 11 D0
630  C074  ORA #;$20        09 20
640  C076  STA;$D011        8D 11 D0
650  C079  JMP;$EA31        4C 31 EA
```

Microtext

Acornsoft
£57.00 cassette

Microtext is one of Acornsoft's more intriguing products. It is described by the company as an **authoring language,** but this means very little to most people.

Essentially Microtext helps anyone teaching with aids and computer-like facilities, without requiring that the teachers have any special computing skills. Typical applications are job training courses, basic school teaching and first level job interviews – although it may come across as rather impersonal on the latter.

The package was designed by the National Physical Laboratory and implemented on the BBC Micro by Ariadne Software. It comes in a standard Acornsoft languages box containing the cassette, a welcome-guide, a reference card, key underlay and a full User Guide. Disc and ROM versions are promised for the future.

The first really noticeable thing about Microtext is its *friendliness.* At all times during command mode typing HELP gives just that. A page of text explaining each of the major commands in reasonable detail. Considering the provision of both a User Guide and the extremely useful laminated quick reference card, a user would find it very difficult to get confused.

At this stage one is naturally tempted to run the suite of example programs provided under the general banner of WELCOME – they are well worth it for introductory purposes, although each has a fairly trivial function.

Once these have been run a thorough read of the manual is essential and fairly lively; the two editors have combined a knowledge of the product with an understanding of others' misunderstandings. Far better than the BBC Micro User Guide itself in fact.

A Microtext program is structured somewhat differently to a conventional program, consisting of *frames* rather than individual lines. Each frame can have a combination of displayed and non-displayed commands. For instance, one may choose to get an input from the subject, and it is fairly likely that a prompt would be issued to guide the subject. In most instances a program would be written in and ultimately use the BBC's Mode 7 (Teletext) display, as this is the most visually appealing and easiest to manipulate. However most other screen modes are supported and more complicated graphics such as charts could then be drawn. In non-teletext modes the red function keys produce foreign characters and mathematical symbols.

Considering that the BBC Micro was originally intended for serious educational purposes, Microtext is a very welcome addition to the Acornsoft catalogue – but why two years after the release of the micro itself? Another case of Acorn's prevarication, joining the second processors, Prestel adaptors and so on. And Microtext does not as yet work properly with the 6502 second processor. **AD**

SOFTWARE FILE

A BUYER'S GUIDE TO UTILITY SOFTWARE

Software file is a monthly rundown on the utility packages available for the BBC, Spectrum, ZX81, Dragon, Oric, CBM 64 and VIC 20 computers. This month the microscope is on the programming languages available in home computing, and on extensions available to standard BASICs supplied with the machine.

MACHINE	PRODUCT	PRICE	FORMAT	MEMORY	SUPPLIER	COMMENT
LANGUAGES – BASIC						
ATARI	MICROSOFT BASIC	59.99	D	32K	ATARI	AN IMPROVEMENT ON THE INADEQUATE ATARI BASIC
ATARI	MICROSOFT BASIC II	59.99	CR	32K	ATARI	CONTAINS MOST OF THE FEATURES OF THE ABOVE
CBM 64	DTL BASIC	35.00 100.00	C D	32K	DATAVIEW	COMPILER – RUNS UP TO 55 TIMES FASTER
CBM 64	SIMON'S BASIC				COMMODORE	
DRAGON	BASIC 09	59.95	D	32K	DRAGON DATA	A STRUCTURED INTERACTIVE COMPILED LANG.
SHARP MZ80	BASIC CYMREIG		C	48K	DAVID COMPUTER'S WARE	WELSH LANGUAGE VERSION OF BASIC
SPECTRUM	BETA BASIC	11.00	C	16K	BETASOFT	AN IMPROVEMENT ON SPECTRUM BASIC WITH NEW KEYWORDS
SPECTRUM	COMPILER	8.00	C	16K	WYE VALLEY SOFTWARE	VERY FAST
SPECTRUM	M CODER II	10.00	C	48K	PERSONAL SOFTWARE	INTEGER BASIC COMPILER
SPECTRUM	SOFCOM	15.00	C	48K	SOFTEK	INTEGER BASIC COMPILER
LANGUAGES – FORTH						
BBC	FORTH	16.85	C/D	32K	ACORNSOFT	1979 STANDARD FORTH
BBC	FORTH	34.72	R	16K	JR BROWN	FIG FORTH
BBC	FORTH	34.00	R	48K	MICRO-AID	OS 0.1/1.0
BBC	FORTH TOOLKIT	10.00	C	16K	LEVEL 9	FORTH EXTENSIONS AND UTILITIES
BBC	LOGO FORTH	59.00	R	32K	HCCS	FORTH BASED LOGO
BBC	RG FORTH	15.00	C	16K	LEVEL 9	1979 FORTH WITH 260 WORDS
BBC	SUPERFORTH	14.95	D	32K	R H ELECTRONICS	FORTH '79 WITH FULL 6502 ASSEMBLER
CBM 64	TINY FORTH	12.95	C/D	UX	ADAMSOFT	MOST ELEMENTS OF FIG FORTH
CBM 64	FORTH	39.00	CR	64K	AUDIOGENIC	FIG FORTH PLUS EXTENSIONS
DRAGON	DRAGON FORTH	19.95	C	32K	DRAGON DATA	NO INFORMATION
DRAGON	FORTH	14.00	D	32K	M & J SOFTWARE	FIG-FORTH + EDITOR/ASSEMBLER
DRAGON	FORTH	5.00	C	32K	M & J SOFTWARE	ENABLES M&J SOFTWARE'S FIG-FORTH TO BE USED WITH PREMIER MICROSYSTEMS DELTA DISK SYSTEM
DRAGON	TELEFORTH	19.95	C	32K	MICRODEAL	
ORIC	FORTH	15.00	C	48K	TANSOFT	INCLUDES EDITOR/ASSEMBLER
SPECTRUM	FORTH	14.95	C	48K	ABERSOFT	FIG-FORTH – AVAILABLE MICRODRIVE
SPECTRUM	FORTH	14.95	C	48K	SINCLAIR	
SPECTRUM	FLOATING POINT FORTH	13.95	C	48K	CP SOFTWARE	FORTH 79 STRUCTURES WITH VARIOUS SPECTRUM FUNCTIONS
LANGUAGES – PASCAL						
BBC	PASCAL T	59.00	R	32K	J.W. BROWN	PASCAL SUBSET ON 16K EPROM SUPPORTED BY EXTENSION DISCS
BBC	S-PASCAL	16.85	C/D	32K	ACORNSOFT	BLOCK STRUCTURED, FULL RECURSIVE SUBSET OF PASCAL
BBC	TINY PASCAL	59.00	R	32K	HCCS	COMPILED PASCAL
CBM64	ZOOM PASCAL	29.95	D	UX	ADAMSOFT	EDITOR, COMPILER AND TRANSLATOR TRU INDEPENDENT 6502 M/C
SPECTRUM	PASCAL 4T	25.00	C	48K	HISOFT	STANDARD PASCAL COMPILER UP TO 50X FASTER THAN BASIC
MISCELLANEOUS LANGUAGES						
BBC	BCPL	99.65	D/R	32K	ACORNSOFT	FULL DEVELOPMENT SYSTEM INC. EDITOR/ASSEMBLER ETC.
BBC	LOGO	8.00	C	32K	SUMMIT SOFTWARE	SIMPLE, STRUCTURED GRAPHICS LANGUAGE
BBC	LISP	16.85	C/D	16K	ACORNSOFT	HIGH LEVEL LANGUAGE INCLUDES M/C INTERPRETER AND UTILITIES
SPECTRUM	LISP 1.3	15.50	C	48K	SERIOUS SOFTWARE	FULL LISP INTERPRETER
SPECTRUM	PROLOG	24.95	C	48K	SINCLAIR	HIGH LEVEL '5th GENERATION-LANGUAGE
SPECTRUM	SCOPE	11.95	C	48K	ISP	STRUCTURED GRAPHICS LANGUAGE
SPECTRUM	SNAILOGO	10.00	C	48K	CP SOFTWARE	

EXCITING ADDITIONS FOR YOUR HOME COMPUTER

KEYBOARD with ELECTRONICS for ZX SPECTRUM

★ Full size, full travel keyboard that simply plugs into expansion port on your Spectrum. ★ Offers single key selection of all major multi-key functions. ★ Extends port for other peripherals. ★ Can accept Atari-type joysticks (optional extra — order 2 of FG66W, £1.36 each and note that case will require cutting).
Three kits needed to build unit: Order LK29G, LK30H & XG35Q. Total price £39.95. Full construction details in Project Book 9 XA09K 70p. Also available ready-built. Order As XG36P. Price £44.95.

KEYBOARD with ELECTRONICS for ZX81

★ Full size, full travel keyboard that's easy to add to your ZX81. ★ No soldering in ZX81; simple instructions make it easy to fit. ★ Makes Shift Lock, Function & Graphics 2 single key selections.
Complete kit (excl. case) LW72P Price £23.95. Case XG17T £4.95. Full construction details in Project Book 3 XA03D. Price 70p. Ready-built in case XG22Y. Price £32.50.

MODEM

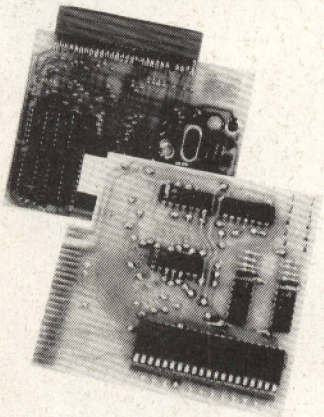
A CCITT standard modem that connects directly to your telephone line via a BT approved transformer. Transmits and receives simultaneously on European standard frequencies at 300 baud. May be used to talk to any other 300 baud European standard modem including the Maplin Computer Shopping modem on 0702 552941 and any British Telecom Datel 200/300 Service modem. The modem's computer interface is RS232 compatible.
Complete kit (excl. case) LW99H. Price £44.95. Case YK62S £9.95. Full construction details in Project Book 5 XA05F Price 70p.

INTERFACES for MODEM

Interfaces are now available for the following machines: Commodore 64, Dragon 32, Oric, Spectrum, VIC20 and ZX81. Each is complete with a Machine Code Communications program. The BBC micro needs no interface and a suitable program is on Maplin catalogue page 15 or Project Book 8, page 59.

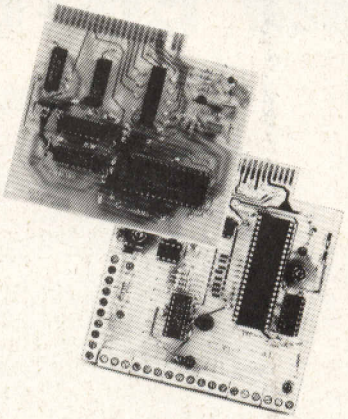
Computer	Order	Details	Price
64/VIC20	LK11M	Book 7	£9.45
Dragon 32	LK12N	Book 8	£18.95
Oric 1	LK40T	Book 10	£13.95
Spectrum	LK21X	Book 8	£19.95
ZX81	LK08J	Book 7	£29.95

Project Book 7 XA07H. Price 70p. Project Book 8 XA08J. Price 70p. Project Book 10. XA10L. Price 70p.



ZX81 I/O PORT

★ Provides two bi-directional ports for 16 input or output lines. ★ One buffered output which can interface directly to CMOS. ★ On board address selection permits expansion to six ports with two boards.
Complete kit LW76H. Price £10.49. Full construction details in Project Book 4 XA04E. Price 70p.



MAPLIN TALK-BACK SPEECH SYNTHESISERS

★ Unlimited vocabulary with allophone (extended phoneme) system. ★ Can be used with unexpanded Oric 1, VIC20 or ZX81 as it does not require large areas of memory. ★ Speech may be easily added to programs. ★In VIC20 version speech output is direct to TV speaker with no additional amplification needed.

Computer	Order	Details	Price
Oric 1	LK28F	Book 9	£23.95
VIC20	LK00A	Book 6	£22.95
ZX81	LK01B	Book 6	£19.95

Project Book 6 XA06G. Price 70p. Project Book 9 XA09K. Price 70p.

DRAGON 32 I/O PORT

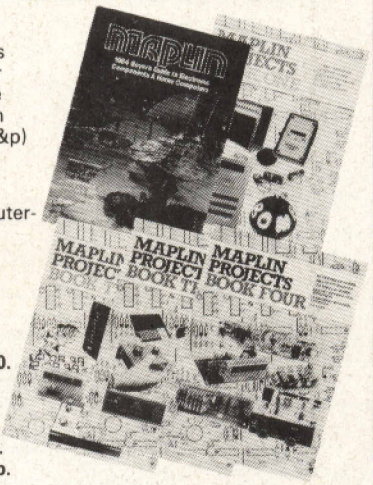
★ Provides two TTL & 3-state bus compatible 8-bit ports, ★ Four norm/inv. latched ports, ★ Two relay switched ports ★ And two opto switched ports. ★ Module plugs directly into cartridge socket and is fully programmable from BASIC.
Complete kit LK18U. Price £14.95. Full construction details in Project Book 8 XA08J. Price 70p.

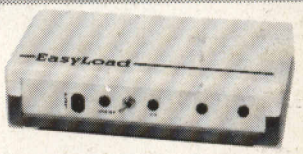
MAPLIN CATALOGUE

Full details of all Maplin's projects and electronic components in our huge 480 page catalogue. On sale now in all branches of W.H. Smith price £1.35. Or send £1.65 (incl. p&p) to our mail order address.

OTHER PROJECTS

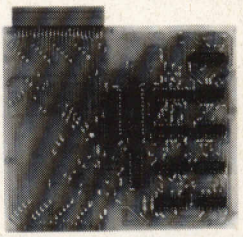
For full details of our other computer-related projects please see the relevant project book as below:
BBC Motherboard — Book 11. ZX81 Sound on TV — Book 6. ZX81 Extendi-RAM — Book 9. VIC20 Extendiboard — Book 9. Dragon 32 Extendiport — Book 10. TTL/RS232 Interface — Book 9. Project Book 6 XA06G. Price 70p. Project Book 9 XA09K. Price 70p. Project Book 10 XA10L. Price 70p. Project Book 11 XA11M. Price 70p.





SPECTRUM EASYLOAD

★ Greatly reduces cassette LOADing & SAVEing problems on Spectrum. ★ Battery powered, no bus connections. ★ Charging from Spectrum PSU. ★ SAVE & LOAD indicators.
Complete kit (excl case) LK39N. Price £9.95. Full construction details in Project Book 10 XA10L. Price 70p.

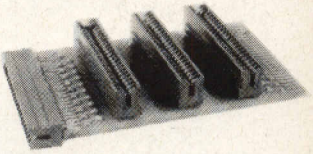


ZX81 HI-RES GRAPHICS

★ Full 256 x 192 fine pixel display with normal or inverted video. ★ Draws lines, circles and triangles, fills and textures. ★ Up to 32 user defined graphics. ★ Operates directly from extended BASIC.
Complete kit LK23A. Price £27.50. Full construction details in Project Book 9 XA09K. Price 70p.

ZX81 SOUNDS GENERATOR

★ Turns your ZX81 into a mini-synthesiser. ★ 3 programmable tone generators. ★ 3 programmable attenuators. ★ Noise generator with 3 pitch levels for special effect sounds. ★ Single address access with PEEK & POKE. ★ Connects directly to extension board or expansion port socket with extra socket (order RK35Q £2.20) ★ Requires separate amp and speaker.
Complete kit LW96E. Price £13.49. Full construction details in Project Book 5 XA05F. Price 70p.



ZX81 EXTENSION BOARD

★Plugs directly into ZX81 expansion port. ★ Accepts a 16K RAM pack and three other plug-in modules simultaneously. Parts are sold separately as follows:
PCB GB08J. Price £2.40. Edge Connectors (4 needed) RK35Q. Price £2.20 each. Track pins (1 pack needed) FL82D. Price 85p per pack of 50.

MAPLIN
ELECTRONIC SUPPLIES LTD

Maplin Electronic Supplies Ltd. Mail Order: P.O. Box 3, Rayleigh, Essex SS6 8LR.
Tel: Southend (0702) 552911. • Shops at: 159-161 King Street, Hammersmith, London W6. Tel: 01-748- 0926.
• 8 Oxford Road, Manchester. Tel: 061-236-0281. • Lynton Square, Perry Bar, Birmingham. Tel: 021-356-7292.
• 282-284 London Road, Westcliff-on-Sea, Essex. Tel: 0702 554000. • 46-48 Bevois Valley Road, Southampton. Tel:0703 25831. All shops closed all day Monday.
All prices include VAT and carriage. Please add 50p handling charge to orders under £5 total (except catalogue).